64572 02 8606

64572 02 8606

CDC 64572

SUPPLY SYSTEMS ANALYSIS SUPERVISOR

(AFSC 64572)

Volume 2

System Software

NEWARK PUBLIC LIBRARY
JUN 3 1986
U. S. GOVT. PUBS.
DEPOSITORY 378

Extension Course Institute
Air University

D 301.26/12-2:

Prepared by
MSgt Roy M. Black Jr.
TSgt Bobby Kazzee

Reviewed by
Carole Cartwright

Edited by
Joyce G. Grimes

3440TH TECHNICAL TRAINING GROUP (ATC)
LOWRY AIR FORCE BASE, COLORADO 80230-5000

EXTENSION COURSE INSTITUTE (AU)
GUNTER AIR FORCE STATION, ALABAMA 36118-5643

Preface

THIS SECOND volume of CDC 64572, *Supply Systems Analysis Supervisor*, is a short course on numbering systems, the purpose and functions of the Executive system of the S1100 computer system, Executive Control Language (ECL), conversational Time Sharing (CTS), Text Editor (ED), and SBSS Software. In Chapter 1, Numbering Systems, you are introduced to binary, octal, decimal, hexadecimal, ASCII, XS-3, and fieldata coding. Chapter 2 contains information on the Executive System. It identifies the purpose and functions, multiple environments, functional areas of the Executive System, and the various system processors used to support the Executive System. Chapter 3 explains the Executive Control Language (ECL) with an explanation of each control statement available. Also covered within this chapter are the data and element processors. Finally, file utility routine/program file utility routine (FURPUR) control statements are also explained. Chapter 4 explains Conversational Time Sharing (CTS) with an explanation of each command available. Chapter 5 covers Text Editor (ED) with explanations of the commands available with this processor. Chapter 6 covers the software which supports the SBSS function. It explains the types of software used in various SBSS programs, file naming conventions, files, and driver complex.

A glossary of terms used in this course is included at the end of this volume.

Code numbers appearing on figures are for preparing agency identification only.

The inclusion of names of any specific commercial product, commodity, or service in this publication is for information purposes only and does not imply indorsement by the Air Force.

To get an *immediate response* to your questions concerning subject matter in this course, call the author at AV 929-4528 between 0800 and 1600 (MT), Monday through Friday. Otherwise, write the author at 3440 TTG/TTMXS-TDB, Lowry AFB CO 80230-5000, to point out technical errors in the text, Volume Review Exercises, or Course Examination. Sending subject matter questions to ECI slows response time.

NOTE: Do not use the Suggestion Program to submit changes to this course.

Consult your education officer, training officer, or NCO if you have questions on course enrollment or administration, Your Key to a Successful Course, and irregularities (possible scoring errors, printing errors, etc.) on the Volume Review Exercises and Course Examination. Send questions these people can't answer to ECI, Gunter AFS AL 36118-5643, on ECI Form 17, Student Request for Assistance.

This volume is valued at 21 hours (7 points).

Material in this volume is reviewed annually for technical accuracy, adequacy, and currency. For SKT purposes the examinee should check the Index of ECI Study Reference Material to determine the correct references to study.

Acknowledgement

PREPARATION OF this volume was aided through the cooperation and courtesy of friends at Keesler AFB, Mississippi, Gunter AFS, Alabama, and the Sperry Corporation. The Sperry Corporation furnished many reference materials used extensively throughout this volume. Permission to use information from Sperry-furnished materials is gratefully acknowledged.

Contents

Chapter		Page
	Preface	*iii*
	Acknowledgement	*iv*
1	Numbering Systems	1
2	Executive System	5
3	Executive Control Language (ECL)	9
4	Conversational Time Sharing (CTS)	23
5	Text Editor (ED)	36
6	SBSS Software	45
	Bibliography	59
	Glossary	60
	Answers for Exercises	61

CHAPTER 1

Numbering Systems

NUMBERING SYSTEMS have been used to account for many things throughout history. Even prehistoric people used a numbering system to count animals they owned. The system probably involved stones or pebbles to represent each unit of their herd. Later, people of ancient China developed written symbols to represent numbers and devised a method of simple arithmetic operations. It wasn't until 700 AD that the Arabs developed the current numbering system. A few hundred years later, European explorers borrowed the mathematical advances from the Arabs for use in navigation. The field of mathematics produced such mathematical geniuses as Sir Isaac Newton (inventor of calculus), Pierre de Fermat (theory of probability), and Albert Einstein (theory of relativity), who applied mathematics to new fields of endeavor. These applications created a new problem: how to work with overwhelming numbers. The abacus (an instrument for performing calculations), used for over 3,000 years, could no longer do the job.

The idea of digital computers was born in 1835 when another mathematical wizard, Charles Babbage, built his "analytical engine." Babbage died in 1871 without perfecting his idea. In 1944, a Harvard professor, Howard Aiken, invented the first working computer. These and other early developments led to the digital computers used today.

Digital computers use numbers in solving problems and recording data. For instance, the Sperry 1100/60 used in the Air Force Automated Supply System is a digital computer. It uses different numbering systems for storage, output, interpretation of balances, dollar values, and descriptive information applicable to each item in the supply inventory.

1-1. S1100/60 Terms

Console operators, system analysts, and programmers of the S1100/60 system must understand four different numbering systems: decimal, binary, octal, and hexadecimal. Their characteristics and uses in the supply computer are explained in this chapter.

Before you can begin a serious study of the S1100/60, you must be familiar with some basic terms. Here, we will review briefly some terms introduced to you in your 3-level studies and then define in a more detailed way the binary, octal, decimal, and hexadecimal numbering systems.

200. Define terms associated with the numbering systems of the S1100/60 system, citing characteristics and uses when necessary.

Base and Radix. These terms refer to the absolute quantity selected as the basis for a number system. This is usually the number of symbols in the system. For example, since there are 10 symbols in the decimal system, its base or radix is 10.

Fixed Value. This term refers to the absolute quantity readily identified with each symbol in a numbering system. For instance, the decimal symbol 3 stands for an absolute quantity of "things," such as 3 dots.

Positional Value. The value obtained by placing additional symbols in front of or after a fixed value is termed "positional value." The fixed value 5 becomes 50 when the symbol 0 is placed after it and .05 if we place a decimal point and the symbol 0 in front of it.

Least Significant. This term refers to the right-most position of a number or character. For example, in the number 50, the zero is in the least significant position.

Most Significant. This term refers to the left-most position of a number or character. In the number 50, the 5 is in the most significant position.

Bit. A binary digit (one symbol), either 0 or 1, is called a bit.

Character. This term refers to one of a set of elementary symbols that may be arranged in ordered groups to express information; these symbols may include the decimal digits 0 through 9, the letters A through Z, punctuation symbols, special input and output symbols, as well as other symbols. Each symbol input to the S1100/60 is stored in 8 binary bits and is called an eight-bit binary character.

Decimal System. The decimal numbering system comes to us naturally since we learned it in early school math classes and use it regularly. We cover it in this section to show the relationship of its characteristics and uses to those of the binary and octal systems. We stated already that the decimal system is a base-10 system that uses these symbols to represent any value: 0,1,2,3,4,5,6,7,8,9. Nine is the largest symbol available in the decimal system. To represent a value higher than the fixed value of nine, combinations of the available symbols must be used and positional values assigned. Symbols are assigned values according to their position in the number. Each position is a multiple (usually a power) of the base and the value of each

THOUSANDS	HUNDREDS	TENS	UNITS	TENTHS	HUNDREDTHS	THOUSANDTHS
5	3	6	7.	1	2	3

GT505908E

Figure 1-1. Decimal positional values.

position remains constant. In the decimal system, the least significant position of a whole number (digits left of the decimal point) is the units, next is tens, then hundreds, thousands, etc. For fractional numbers (the digits right of the decimal point), positional values are tenths, hundredths, thousandths, etc. For an example of the positional values in the decimal system to the left and right of the decimal point, see figure 1-1.

Binary Systems. All basic computer functions can be expressed in a two-state system. Thus, a computer needs only the 0 and 1 to represent any value. As explained earlier, the value of these symbols depends on their position. The least significant position of a binary number has the positional value of one. Each position from least to most significant is two times greater than the previous position, and each position represents a power of 2. This concept should be clearer to you when you study figure 1-2 for binary positional values and related powers of two.

Octal System. The word "octal" is a derivative of the Latin word "octo" meaning eight. The octal numbering system is a base-8 system using these symbols 0,1,2,3,4,5,6,7. Except for eights and nines, the symbols used are the same as those used in the decimal system. This requires special identification of numbers as octal. Octal numbers are identified by saying "octal 5362," by writing the number preceded by a zero 0562, or by using a subscript 5632_8.

The use of octal on the S1100/60 makes the reading of binary numbers easier for programmers and operators. Using this system, you can read three binary digits as one octal number. You just need to keep in mind that 8 is a power of 2 (2^3). Refer to figure 1-3 to compare the octal and binary systems.

It is far easier to read a string of binary numbers in octal than it is to read them in binary. For example, to read binary 101101010, you would say, "one, zero, one, one, etc." In octal, you merely say "octal five-five-two."

Hexidecimal System. The hexidecimal number system is ideally formatted for computer systems using 8 bit characters. This system is a base-16 system using these symbols: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F. Positional values of the hexadecimal digits increase to the left of the least significant digit by the powers of 16 (fig. 1-4). Using the hexidecimal system (as with the octal system) makes the reading of binary numbers easier for programmers or operators. You can read four binary digits as one hexadecimal number.

BINARY POSITION	2^5	2^4	2^3	2^2	2^1	2^0	
POSITIONAL VALUE	2 x 2 = 4 x 2 = 8 x 2 = 16 x 2 = 32	16	8	4	2	1	
EXAMPLE BINARY NUMBER	0	1	1	0	0	1	
DECIMAL VALUE	0	16	8	0	0	1	= 25

505908F2-80

Figure 1-2. Binary positional values.

OCTAL	BINARY	OCTAL	BINARY
0	000	4	100
1	001	5	101
2	010	6	110
3	011	7	111

505908C1-80

Figure 1–3. Octal/binary comparisons.

Exercises (200):

1. Define "fixed value."
2. What term refers to the left-most position of a number?
3. What is a "character" in mathematics?
4. What term refers to the right-most position of a number?
5. State the four numbering systems, giving the base and symbols used in each system.

BINARY	DECIMAL	HEXADECIMAL
0000	0	0
0001	1	1
0010	2	2
0110	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

505908C2-80

Figure 1–4. Hexadecimal positional values.

6. Briefly state the uses of the binary, octal, decimal, and hexadecimal systems in the S1100/60.

7. How many binary bits are represented in a hexadecimal digit?

8. By what powers does the positional value of the hexidecimal numbering system increase?

1-2. S1100/60 Codes

When Charles Babbage created his "analytical engine" in 1835, he used punched card input. An engineer named Jacquard had already invented punched cards to use with his weaving loom. In 1894, a census bureau mathematician named Herman Hollerith revolutionized the punched card system with his invention of the Hollerith code. As computers have become more advanced and data communication systems have emerged, cards are being used less and less. Other forms of data notation and representation have come to be used.

201. Cite the characteristics and uses of character codes used on the S1100/60 system.

American Standard Code for Information Interchange (ASCII). ASCII is used throughout the world to represent internal computer data. It allows the user readily to relate and understand data coded in binary form. ASCII is an 8-bit code providing 256 characters, 0–9, A–Z, a–z, and various special characters.

Fieldata. Fieldata is a code used in data fields in the S1100/60 system. It consists of uppercase letters A–Z, 0–9, and special characters. Fieldata may be used instead of ASCII for processing conversational Time Sharing (CTS) and other languages.

XS–3. XS–3 coding was developed for use in the old U1050–II computer system. As numeric and alphabetic characters entered the computer, they were converted to a pure 6-bit binary character. A binary three was then added to the pure binary to convert it to the U1050–II machine code; hence, the name XS–3. XS–3 is still used on certain flags (other asset flag, data control flag, program control flag, and type level detail flag). In fact you may need to interpret the bit structure of these flags when you develop SURGE programs.

Exercises (201):

1. What does ASCII stand for?

2. How many characters may be represented in ASCII?

3. What characters may be represented in fieldata?

4. What flags still use the XS–3 notation?

Executive System

THE PROCESSING environment of the S1100/60 Computer System consists of an operating system that controls the hardware and the executive system, or EXEC, that controls the user programs and files needed for processing. Let's look at how the EXEC handles the processing of a report.

Major Jones, in stock control, has requested an R32. You prepare the runstream and start the R32. Already the EXEC has let you start processing the R32. The EXEC will, in turn, decide (1) how much disk and memory space, (2) which hardware devices, and (3) which files you need for processing your job. Some of the files that the EXEC assigns are USAF*GVABSUD001., USAF*GVECLUD001., 2BPS001*GV801AUD001., ITMDTL area, and the output file. Some of the hardware devices assigned could be the RPS, disk drive, and tape drive.

As you can see, the EXEC is quite complex. In this chapter we will discuss the EXEC, and then break it down so that you can understand how it works.

2-1. Executive System

To ensure smooth and efficient operations of the S1100/60, the operating system and S1100 Executive work hand in hand. The operating system handles the basic programming for the S1100/60. The S1100 executive handles the processing of jobs, from beginning to end, in software operations in conjunction with the operating system.

202. State the purpose of the Executive System and define its major functions.

In a large computer system, it is necessary to have a controller—a high-level program that maintains the overall system environment. On the S1100/60 the controller is the EXEC. The EXEC allows concurrent operations of many programs—reacting to the inquiries, requests, and demands of its many different users. It can sort, file, retrieve, and protect large blocks of data. It also makes the best use of available hardware facilities and attempts to minimize job time. The EXEC is quite complex, consisting of over 400 routines which perform three functions: runstream processing, resource/facilities allocation, and input/output handling.

Runstream Processing. The primary function of the EXEC is to process runstreams. The EXEC performs edits to verify the format of the runstreams and the control statements within. Once the runstream is accepted and all edits are passed, the runstream is processed to completion.

Resource/Facilities Allocation. The EXEC also manages all system resources (tape, disk drives, files, etc.,), and allocates them to individual runs. To process a request for a run, the EXEC first allocates all required resources. These resources include memory, space, central processor time, peripheral devices, and mass storage space, including space for temporary work files.

Input/Output (I/O) Handling. The EXEC performs all I/O operations for user programs. User programs are not allowed to execute hardware I/O instructions. Due to the sensitive nature of I/O operations and the relatively slow speed of peripheral devices, all I/O operations are handled by the EXEC. This provides for faster, more efficient processing.

Exercises (202):

1. What is the purpose of the EXEC?

2. List and define the major functions of the EXEC.

2-2. Multiple Environments

The capabilities of the EXEC cover a great variety of data processing activities. Its design offers versatility to handle batch, demand, and transaction processing and provides multiprocessing/multiprogramming capabilities. As with most computer systems of its generation, the S1100/60 offers the user the capability to process in a multiple processing environment. This section will explore the different environments the S1100/60 offers the user.

203. Identify characteristics of the processing environments of the EXEC.

Batch. A batch run is submitted to the EXEC as an entire job, usually in the form of a card deck or disk file. The EXEC processes the batch run from start to finish with little or no user interaction. When a batch run is entered into the system, it is read by the EXEC and placed into a system

backlog queue. Each batch run remains in the queue until it becomes the highest priority and all the resources required are available for allocation. Reports are produced during batch operations.

Demand. A demand run is a job in which you issue a single request (control statement) at a time through a demand terminal. Each request is processed immediately. Demand jobs run at a higher priority than batch. A few examples of demand runs are working in CTS, Text Editor, or with any processor on the system. The demand environment allows the user to interact directly with the EXEC.

Transaction. In transaction processing, predefined programs are called by the EXEC to process supply transaction. Because transaction runs are the highest priority on the system, they normally run in a very short amount of time. Most of your daily online activities are transaction processing.

Multiprogramming and multiprocessing. Runs may come from two major sources: remote and central. These various runs, through the EXEC's use and control of efficient multiprogramming and multiprocessing techniques, may, at any given time, be in different stages of activity. Input and output processing may occur at the same time. For this reason, you need to understand the characteristics of the multiprogramming and multiprocessing environments.

Multiprogramming. This environment can be defined as the ability to process many programs in the same time frame. Although the programs are executed one instruction at a time, it seems that all programs are being processed simultaneously. The S1100/60 uses the central processing unit (CPU) time efficiently by not waiting between runs.

Multiprocessing. The multiprocessing environment involves the use of coupled CPUs. The scheduling and CPU switching techniques used in the EXEC provide the multiprocessing capabilities. The extension to multiprocessing leads naturally from the multiprogramming aspects of the system where many independent activities in user programs are available for execution. The difference between multi*programming* and multi*processing* is that in multiprogramming the system only seems to execute many instructions simultaneously, whereas in multiprocessing the system actually is able to process multiple instructions. Its capability, of course, depends on the number of CPUs coupled to the system.

Security. The security environment within the EXEC ensures that the S1100 Operating System is a controlled access system. Security in the S1100 restricts the assignment and access to computer resources and data to specifically authorized objects, such as users, cataloged files, devices, terminals, and programs. This capability ensures that a given user may only process against its own files and data base.

In this section we have discussed many environments of the EXEC and the S1100/60 system. Please do not think that each is independent or separate from the other. The EXEC can do batch, demand, and transaction processing simultaneously through multiprogramming or multiprocessing. Keep in mind security is active while processing in any mode.

Exercises (203):

Match each item in column B with the appropriate item in column A. Each response in column B may be used once, more than once, or not at all.

Column A	*Column B*
____ (1) Process reports.	a. Batch.
____ (2) Controls access to the operating system.	b. Demand.
____ (3) Process using couped CPUs.	c. Transaction processing.
____ (4) Process many jobs in the same time frame, using one CPU.	d. Multiprocessing.
____ (5) Execute predefined programs based on supply transactions.	e. Multiprogramming.
____ (6) Process a CTS control statement, awaiting immediate response.	f. Security.
____ (7) Process using backlog queues.	

2-3. Functional Areas of the Executive System.

As stated in the previous section, the EXEC is responsible for the efficient operation of the S1100 environment. To perform this function, the EXEC depends on the supervisor for scheduling runs for processing. The EXEC is also responsible for facilities assignment, file control, operator communications, and controlling the I/O devices. In this section we will attempt to explore the different functional areas of the EXEC and how they are used.

204. Cite the purpose of the EXEC's primary functional areas.

Supervisor. The supervisor is the EXEC component that controls sequencing, setup, and execution of all runs. It controls the execution of many independent and interdependent programs. The supervisor has three levels of scheduling: coarse scheduling, dynamic allocation of storage space, and CPU dispatching. Runs entering the system are sorted in information files used by the supervisor for run scheduling and processing. Control statements for each run are retrieved and scanned by a control statement interpreter in the supervisor to help and speed the selection of runs for setup by the coarse scheduler.

The coarse scheduler schedules runs based on two factors: the priority of the run and its facility requirements. The priority ranges from the letters A to Z, with A being the highest priority. The user or runstream initiator is free to assign any priority to a run. However, the individual must have permission from the DPI to deviate from the priority assigned to the RPS site. The default is "M" if a priority is not requested on the '@RUN' control statement. Within a given priority, the selection for opening a run from backlog is essentially on a first-in/first-out basis. In reality, there are other factors that affect priority, such as the amount of processing time for a run. The coarse scheduler makes priority considerations for opening runs. When a batch run is presented to the EXEC, it is usually placed in the system's backlog queue, where it remains until the EXEC decides sufficient system resources exist to execute the

runs. All runs placed in backlog have a priority. When the EXEC decides to open another run, it selects the highest priority run from backlog.

The dynamic allocator takes runs set up by the coarse scheduler and allots main storage space according to the needs of the individual tasks (programs) for each run. Normally, tasks from many different runs are in main storage at the same time. Each run may be thought of as being made up of tasks, where a task is a single operation of a system processor or the execution of a user program. All tasks for a given run are processed serially but not necessarily consecutively. If there are several runs, the tasks of separate runs are interleaved. When time sharing of main storage is appropriate (for example, the total main storage requirements of all programs in a state ready for execution are greater than the available main storage), the dynamic allocator initiates storage swaps. This involves writing one or more programs from main storage to mass storage and allocating the now available main storage to other runs. Such action provides reasonable response time to remote demand processing terminals or to satisfy batch priority requirements.

The CPU dispatching routine is the third level of scheduling. It switches the commitment of the CPU from one task to another. Under normal circumstances, a batch program is allowed to use a CPU either until it is interlocked against some event or some higher priority program is freed of all of its interlocks. On multiprocessor systems, two or more tasks are executed at the same time.

Facilities Assignment. Available facilities (drum, disk, tape, etc.,) and their dispositions are indicated to the system at system generation. Thereafter, the EXEC assigns these facilities, as needed and as available, to fulfill the facilities requirements of all runs. The EXEC maintains current tables of the system's facilities showing those assigned to a run or available for assignment. If a requested facility is not available, the run is placed in a hold state and information pertinent to the run and the reason for the hold are stored in a queue. A held run remains in the queue until the facilities are available. At this time the run is released.

File Control. The executive file control routines afford the highest degree of operation flexibility in storing and retrieving data without concern for the physical characteristics of the recording devices. Security measures ensure that files are not subject to unauthorized use or destruction. File control routines are provided to roll out files from mass storage devices to magnetic tape, as well as to reconstruct such files on the mass storage devices when the user calls for them.

Comprehensive utility routines are available for manipulation of files and for informing the user of current status and structure of files. Provisions are made for random storage and retrieval of data under the direction of the user. User program files and data files are maintained and processed in the same environment.

Operator Communications. Operator functions are required for a large variety of activities. The EXEC groups them into classes, allowing a division of operator duties in a multioperator installation. These functions fall into four broad categories: system control, input/output, communications, and hardware confidence activities. These functions may be associated with one or more system consoles depending on the complexity and layout of the installation. Also, remote terminals may be configured and used as system consoles. The capabilities of these remote consoles may range from very limited to almost those of an onsite console (as are the capabilities of the RPS console, function number 057).

The EXEC displays information such as current system load and operator requests associated with I/O setup and interlocks. The operator can request other information, such as backlog status. If the display area becomes full, the Executive defers lower priority displays.

Input/Output (I/O) Device Handlers and Symbionts. The input/output device handlers and symbionts control the activities of all I/O channels and peripheral equipment attached to the S1100/60.

The input/output control routine consists of two parts: the initiator and interrupt answerer. The initiator checks the status of hardware, selects the processing path, sends requests to the appropriate queue, and issues the I/O request to the specified device. The interrupt answerer handles the interrupt or error condition, initial validation of the interrupt, queues the request (either error or normal completion) for postprocessing, and issues the next I/O request.

The EXEC uses its multiprogramming/multiprocessing capabilities to divorce the executing programs from the slow-speed peripherals. All input from and output to peripherals (card readers, printers, punches, etc.,) are buffered through high-speed mass storage devices. This is provided by a set of routines called symbionts. Symbionts operate independently of all other programs and copy data between the slow speed peripherals (hence called symbiont devices) and high-speed devices.

Exercises (204):

1. Explain the purpose and role of selective functions of the EXEC.

   a. File Control.

   b. Facilities Assignment.

2. What are the four categories of operator communications?

3. List and explain the supervisor's three levels of scheduling.

4. List and explain the two parts of the I/O control routine.

2-4. System Processors

The system processors are programs that operate on the S1100/60 as an extension of the EXEC. These programs are required by the EXEC to maintain the operation of the system. They perform functions to create, modify, and maintain files, produce backup file copies, and provide diagnostic information on the status of the system and programs upon termination. We will discuss a few of these system processors in this section.

205. Differentiate among system processors associated with the S1100/60 system.

File Utility Routine/Program File Utility Routine (FURPUR). FURPUR consists of a set of file maintenance routines that provide flexibility in managing and manipulating cataloged or temporary files. Through FURPUR the user can copy files, delete files, and even change file characteristics. FURPURs are covered more in depth later in this volume.

Postmortem Dump Processor (PMD). The PMD produces edited dumps of the contents of main storage at program termination. The PMD provides users and programmers a research tool when a run is ERROR FIN'ed. The PMD is called by the '@PMD' control statement. A diagnostic file (DIAG$), used for recording diagnostic information for the PMD, is assigned automatically to each run.

The PMD offers the user many options and various formats to help the user debug runstreams and programs. The user may request specific dumps at the main storage area for named elements, segments, or banks. Dumps can be produced in Octal, ASCII or Fieldata formats. Without the PMD it is virtually impossible to diagnose error conditions and software problems. Additional information can be found in UP–8730, Sperry Univac Series 1100 Series System Utilities Programmer Reference.

Data Processor. The data processor inserts, adds, or modifies data in data files. The data processor inserts symbolic images or applies updates to a data file, thus generating a second data file reflecting those corrections. The data processor is called by the '@DATA' statement. The data processor control statement is covered more in depth later in this volume.

Element Processor. The element processor introduces an element into a program file or makes changes to a symbolic element in a program file. The element or correction images immediately follow the '@ELT' call in the ELT processor. These images are terminated by another control command. The element processor is covered further in this volume.

File Administration Processor (SECURE). The secure processor provides the ability to copy files to tape for backup purposes and to restore files to disk from tape when necessary. The ARCHIVE command produces backup copies and deletes files. The LOAD command copies files from backup tape to mass storage. The SAVEALL command initiates a backup tape for all files on the system. The LIST FILES command provides a summary listing of files loaded on the S1100/60; however, this is done through coordination with the DPI.

Sort/Merge Processors (SORT). The S1100 sort/merge package is fully modular with every functional unit completely self-contained. This permits the various units to individually adapt to their own particular task, enabling them the most effective form, allowing updating and augmentation. In the internal sort, the replacement selection method is used, which takes advantage of any inherent sequence in the original data. The input data can be sorted on magnetic tape, punched cards, or mass storage. The sort/merge processor can handle any collating sequence (fixed or variable) specified by the user.

Exercises (205):

1. What processor introduces an element into a particular program file?

2. What is the purpose of the postmortem dump processor?

3. What command of the secure processor allows the user to create a backup tape file for disk files and then deletes the disk file?

4. What collating sequence may the sort/merge processor handle?

CHAPTER 3

Executive Control Language (ECL)

THE EXECUTIVE Control Language (ECL) is a set of statements with which users (programmers, operators, etc.,) communicate with the EXEC. These statements must be in prescribed formats. In this chapter we will cover various ECL statements, their uses, and formats.

3-1. Basic Control Statement Format

Before we begin the discussion of ECL control statements, it is important for you to understand the *basic format* of the control statement. In this section we will discuss the various fields and parameters of the control statement, rules of annotation, and continuation.

206. Define the fields associated with the basic ECL control statement and cite the characteristics of transparent control statements.

The control statement, with the exception of transparent control statements, consists of a recognition character (@) and five major sections:

(1) The label field - @[label:].
(2) The operation field - the command.
(3) The option field - the options for the command.
(4) The operand fields - parameters.
(5) Comments.

Label. The label field need not appear, but it may be used to name a control statement. The label is limited to six alphanumeric characters. The first character must be alphabetic. If a label is specified, it is followed immediately by the colon (:).

Operations. The operations field contains the command to be executed. The operations field is terminated by a blank, or comma if options are specified.

Options. The options field specifies the options in the form of unsequenced alphabetic characters related to the particular command. A blank character separates the option field from the operand field.

Operand. The operand fields specify parameters associated with the command field. The content of each operand field, the number of operand fields and whether each is required or optional varies with the command selected.

Comment. The comment field contains any comments the user provides. The semicolon (;) cannot be used in the comment field.

Control Statement Continuation. In certain situations, a control statement may require more than one line. To continue the control statement, you code the end of the first line with a semicolon (;). Continuation of the next line can begin in any column, with one exception: the @ character must not be placed in column one of the continuation line.

Leading Blanks. Leading blanks within a statement are permissible only in the following cases:

(1) Following the @ character.
(2) Following a colon (:) when a label is specified.
(3) Following a parameter subfield separator (/).
(4) Within the comment field.

A blank placed at any position in the coding other than those listed identifies the end of the control statement.

Transparent Control Statements. Transparent control statements are identified by the two @ characters preceding the statement (for example, @@TERM). Transparent control statements may be processed during demand mode processing. These statements are processed immediately after input from a terminal. The processing called for by the control statement is done independently of any current program execution. Transparent control statements are a subset of the control statement set. The syntax rules for normal control statements apply.

In this section, we have covered the basic format of ECL control statements. We also identified control statement continuation, annotation, and leading blanks. We discussed the subset of ECL control statements, their use, and difference in syntax.

Exercises (206):

1. List the fields associated with the basic control statement format.

2. What field contains the command of the control statement?

3. What field is followed by the colon (:)?

4. What field specifies parameters associated with the command field?

5. What character cannot be used within the comment field?

6. What character separates the option field from the operand field?

7. How are transparent control statements identified?

3-2. Filenaming Conventions

If a program creates a file, the file format is determined by the program. The programmer determines the number of records, the record length, the logical elements, the access methods, and the file organization. This type of file is referred to as a user-formatted file. Other files are generated, formatted, and controlled by the data base administrator. In this section we'll discuss a third category of files—those determined by the EXEC or system processors. These files must conform to one of the three types of formats expected by the processors: data, program, and element. For referencing and identification purposes, these files must have names. In this section we will discuss the rules and parameters that make up the filenaming conventions of the S1100/60 system.

207. State general principles and identify elements of filenaming conventions.

Each file in the operating system is assigned a unique name to distinguish it from all other files. The filename is required in the control statements that reference files. The filename is made up of the following elements:
[qualifier*]filename[(f-cycle)][/read-key][/write-key].

Qualifier. The qualifier establishes uniqueness between files that have the same basic filename. The qualifier consists of 1 to 12 alphanumeric characters, -, and $.

Basic Filename. This field identifies the basic filename of the file. It consists of 1 to 12 characters, -, and $. If omitted, the default filename is the RUNID of the run.

F-cycle. The F-cycle identifies a particular file from a set of related files that have the same qualifier and filename. The F-cycle is assigned by the system or may be modified by you the user. There are two types of F-cycles: relative and absolute.

As a file of a given F-cycle set is created, a number is assigned to it. This number is called the absolute F-cycle number and it uniquely identifies that particular file within its F-cycle set. Absolute F-cycle numbers begin with the number 1 and continue through 999, at which point the numbering recycles back to 1. A maximum of 32 consecutively numbered files may be retained in a given set.

Another way to reference files within a given F-cycle set is through the use of the relative F-cycle number. Relative F-cycle numbers are integers in the range of +1 to −31. Relative F-cycle number 0 refers to the cataloged file whose absolute F-cycle number is the highest. Relative F-cycle number −1 refers to the next highest. The highest is defined as having the highest absolute F-cycle number of any of the files in the F-cycle set unless the absolute F-cycle numbers are at a point of recycling from 999 to 1, in which case 1 is higher than 999.

To create the next sequential absolute F-cycle, catalogue a file using the relative "+1".

Read and Write Keys. The read key and write key provide security for a file. Each consists of 1 to 6 characters except for blanks, commas, slashes, periods, and semicolons.

The period identifies the end of the filename.

When naming a file, all parameters are optional except the basic filename. For those parameters omitted, the executive provides a standard default. If the qualifier and the * are omitted, the project-id from the @RUN control statement is used as the qualifier. If the read key or write key is omitted, blanks are supplied as the key. If the F-cycle is omitted, the relative cycle number "−0" is supplied. If the qualifier is not specified for a temporary file, then the qualifier 'TPF$' is assumed.

External and Internal Filenames. The term "external filename" refers to the name of a cataloged file or a temporary file assigned to a particular run. As previously stated, it may be necessary to specify the qualifier, filename, and F-cycle to ensure the file is uniquely identified. It may be convenient to have a short name to refer to a file that has a long external filename. By means of the @USE control statement, the executive attaches an alternate or internal filename to a file for referencing within a run. Internal filenames may consist of 1 to 12 alphanumeric characters, -, and $. For example the filename EZ is the internal filename for file ABCDEFGHIJKL*LKJIHGFEDCBA(−23). by the control statement: @USE EZ,ABCDEFGHIJKL*LKJIHGFEDCBA(−23). For the remainder of the run, whenever a reference to the file is made, EZ can be used as the filename. Thus the file can be referenced by the external or internal filename.

Element Names. There are several types of elements recognized by the operating system. The four most common are symbolic elements, relocatable elements, absolute, and omnibus. Symbolic elements contain source images for language or system processors. Relocatable elements are the output of certain processors such as COBOL and FORTRAN. Absolute elements are the final output of a compile program. Omnibus elements are arbitrary format elements that store any kind of information.

Elements of any types are maintained in program files on mass storage or in element files on magnetic tape. Each element is uniquely identified within a file by its element

type, name, and version. The following notation specifies element names:
[filename].element-name[/version][(S-cycle)]. The filename is the name of the program file for that element. The element-name is the name of the element. The version establishes uniqueness between elements in the same file that have the same type and element-name. The S-cycle specifies a particular version of a symbolic element; S-cycles are only used for symbolic elements.

The S1100/60 allows five consecutively numbered S-cycles of a symbolic element. As with F-cycle files, elements may be referenced by the absolute and relative cycle numbers. The maximum absolute S-cycle number is 62. Absolute element cycles are numbered from 0 to 62.

The element-name and version may consist of 1 to 12 alphanumeric characters, -, and $. The S-cycle is an integer number. All parameters except element-name are optional when naming an element.

Exercises (207):

1. State the components that make up a filename.

2. What is the purpose of the qualifier?

3. What is the difference between "external" and "internal" filenames?

4. What is the purpose of file-cycles (F-cycles) and element-cycles?

5. Name the components of the element name.

6. Name four types of elements.

3-3. ECL Control Statements

ECL Control Statements provide the heart of an efficient and effective S1100/60 operation. You may be required at times to design, build, and analyze runstreams consisting of ECL Control Statements. You must understand the types of ECL Control Statements and know how they are used. This section will discuss ECL Control Statements you need to be familiar with. For discussion, they are divided into functional groupings: scheduling, facilities, message, symbiont directive, and dynamic runstream modification.

208. Associate functions with ECL scheduling control statements.

Scheduling Control Statements. Scheduling control statements identify the beginning and ending of each runstream to the EXEC. They also control the processing of each runstream.

Run initiation (@RUN). This control statement (fig. 3-1) identifies the run to the EXEC and provides the information needed for accounting and scheduling purposes. The @RUN must be the first control statement in the runstream. Accounting information is provided by the use of the RUN–ID, ACCT–ID, and PROJECT–ID fields of the @RUN control statement. They identify the name of the RUN and who to charge with the CPU time. This information is accumulated and displayed at RUN termination.

Scheduling is based on priority, options, and the remaining parameters on the control statement. The priority, although set by the @RUN control statement, may be manipulated by the S1100/60 console operator. This may prove necessary to remove bottlenecks in the system and ensure smooth and efficient processing of the system.

There are various options available for the @RUN control statement. The most commonly used are C, P, and R. The 'C' and 'P' options, when used with the "pages/cards" parameters, provide the capability to terminate the runstream if the estimated number of cards and/or pages is exceeded. These are useful options when testing a SURGE or QLP retrievals. The 'R' option restarts the run in the event of a recoverable system failure.The other parameters of the @RUN statement also provide scheduling information. The START–TIME and DEADLINE parameters specify the time required to start the run and to terminate it. The run-time parameter ('T' option) terminates the run if the estimated CPU usage time is exceeded.

The example of a @RUN control statement in figure 3-1 identifies the run for run-id QLP. The account-id is SBSS; the user-id is LGSS. The project-id is 2USAF. Notice that no priority is designated. This establishes a default priority of 'M' for the run. The slash (/) is necessary to identify the option field. In this run the option is 'P'. This option causes the run to terminate when the estimated number of pages output for the run has been exceeded. We can see by the "pages/cards" parameter of the @RUN control statement that we have estimated 50 pages for this run. Therefore, the run 'QLP' will terminate if 50 pages is exceeded by the output.

Run termination (@FIN). This control statement identifies the end of a run. The @FIN control statement must appear as the last statement in all runs. When the @FIN control statement is encountered, the accounting routines are called and all facilities assigned to the run are released. The operating system prepares a printout summary that includes initiation time, termination time, page count of printed output, and all operator's console messages. The @FIN control statement cannot be continued and must be entered in the first four columns of the input image.

Dynamic initiation of an independent run (@START). The @START control statement permits the user to schedule independent batch runs where the runstreams for these runs have been previously created and entered into the system. This feature allows the user to automatically schedule run execution at a time of choosing (for example, on a daily basis).

The example of the @START control statement as shown in figure 3-1 starts the run contained in element UTL068 of file SBSS*REPORTS..

Initiating execution (@XQT). The @XQT control statement initiates the execution of an absolute element. This control statement is necessary for all SBSS programs processing under the control of program GV801A.

The example in figure 3-1 shows an @XQT control statement initiating the execution of the absolute element called GV801A in program file USAF*GVABSUD001.. The 'S' option directs the execution of GV801A to the secondary gang for concurrent processing.

Exercises (208):

1. What scheduling control statement identifies the run to the executive?

2. What scheduling control statement identifies the end of a run?

3. What scheduling control statement initiates the execution of an absolute element?

4. What scheduling control statement permits the user to schedule independent batch runs?

5. Write the control statement to identify the run 'D02' for project-id 1USAF, user-id LGSS, and account-id SBSS. Assign a priority of 'M' and assign the start time of the run at 1800 hrs.

6. Write the control statement that will initiate execution of the absolute element PROG contained in the program file USAF*GVABSUD001.

209. Identify the function and the options of specified ECL facilities control statements.

Facilities Control Statements. Facilities control statements catalog, assign, delete, and release files and specify the facility requirements for files used in runstreams. See figure 3-2 for formats of the control statements discussed.

Assigning files and peripheral devices (@ASG). The @ASG control statement names a file, states I/O facility requirements, and assigns it to the requesting run. If the file is cataloged, the facility requirements are known and are specified when assigning the file. The options available with the @ASG control statement are numerous, from A to Z. The five most commonly used options are A,T,C,U, and P. The 'A' option assigns an existing file to the run. The 'T' option assigns a temporary file. The C and U options catalog a file after the run terminates. The 'C' option catalogs file on the master file directory but only if the run terminates normally or if an @FREE control statement releases the file. The 'U' option is similar, except that the file is cataloged regardless of run termination. The 'P' option catalogs a public file. If the 'P' option is omitted, the file is cataloged as a private file. It can only be accessed by runs having the same project-id or acct-id as the run that created the file.

Figure 3-2 provides an example of the @ASG cataloging (U) and assigning a public file (P) called SBSS*Reports. to the run. This file is cataloged on the master file directory regardless of how the run terminates.

Cataloging of files (@CAT). The @CAT control statement catalogs a file without having it assigned to the run. The @CAT is preferred to the @ASG as it ensures the file is created prior to writing data into the file. One of the most used options of the @CAT is the 'P' option. This has the same use as with the @ASG control statement. It is important to remember that if you don't specify this option, the file is cataloged as a private file. To access this file from a demand terminal, you must log on in the exact way you did when you cataloged the file. You must also use the same user-id/password. If you don't do this, you cannot access this file. Normally files are cataloged as public to prevent this problem.

The example in figure 3-2 shows the way files are normally cataloged and assigned in a run. Notice that the file is first cataloged as public (P), and then assigned to the run with the @ASG control statement.

Releasing files and peripheral devices (@FREE). The @FREE control statement releases files and their facilities (disk or tape) and removes the exclusive use of areas assigned to the run. A file that is released by a @FREE control statement can no longer be referenced by the run. It can be reassigned by an @ASG control statement. As a rule of thumb, files and facilities should be released when they are no longer needed. If they are not released by the means of the @FREE control statement, they are retained until the run is terminated by the @FIN control statement. Therefore, if you have a rather involved run or are using a demand terminal, make it a practice to release files when they are no longer used.

Some of the options available for the @FREE control statement are D, I, and X. The 'D' option decatalogs the filename specified by the filename parameters. The 'I' option inhibits the cataloging of the file if it was assigned to the run by the use of the 'C' or 'U' options of the @ASG control statement.

```
@RUN[,Priority/Options] run-id,acct-id/userid,project-id,
runtime/deadline,pages/cards,start-time

Example:  @RUN,/P  QLP,SBSS/LGSS,2USAF,,50

@FIN

@[LABEL:]START  runstream-filename

Example:  @START  SBSS*REPORTS.UTL068

@[LABEL:]XQT[,Options] [filename.]element-name

Example:  @XQT,S USAF*GVABSUD001.GV801A
```

Figure 3–1. Scheduling control statements.

```
@[LABEL:]ASG[,Options] filename

Example:  @ASG,UP  SBSS*REPORTS.

@[LABEL:]CAT[,Options]  filename

Example:  @CAT,P SBSS*REPORTS.
          @ASG,A SBSS*REPORTS.

@[LABEL:]FREE[,Options] filename-1,filename-2,...,filename-n

Example:  @FREE  SBSS*REPORTS.

@[LABEL:]USE  internal-filename,external-filename.

Example:  @USE A,SBSS*REPORTS.

@[LABEL:]QUAL  qualifier

Example:  @USE SBSS
```

Figure 3–2. Facilities control statements.

Finally the 'X' option removes the exclusive use of the file assigned to the run.

The example in figure 3-2 releases the file SBSS*REPORTS. and its facilities from the run.

Attaching internal filenames (@USE). The @USE control statement equates internal filenames to external filenames. Once this is done, any file can be referenced by its internal or external filename.

In figure 3-2 the internal filename for the file SBSS*REPORTS. is 'A'. From the point the @USE control statement is processed in the run until it is either changed, FREE'd, or the run terminates, the file SBSS*REPORTS. may be referred to as 'A'.

Specifying the filename qualifier (@QUAL). The @QUAL control statement specifies a standard qualifier for implied usage on succeeding control statements involving filenames.

This control statement is useful when accessing many files under a certain qualifier. Now instead of specifying the qualifier every time you need to access the files, you may imply the new qualifier through the @QUAL control statement. After this is accomplished, you need only to precede the filename by an asterisk (*) to use the implied qualifier identified by the last @QUAL control statement.

The example in figure 3-2 identifies 'SBSS' as the implied qualifier. From this point on or until another @QUAL control statement is encountered, the file SBSS*REPORTS. may be accessed by annotating the file as '*REPORTS.'.

Exercises (209):

1. What does the @ASG facility control statement accomplish?

2. What is the function of the @CAT facility control statement?

3. What is the function of the @FREE facility control statement?

4. What is the function of the @USE facility control statement?

5. What is the function of the @QUAL facility control statement?

6. Write the control statement that will assign the previously cataloged file USAF*GVABSUD001. to the run.

7. Write the control statement that will catalog the file SBSS*RUNSTREAM. as a public file with a write key of ETIRW.

210. Relate functions to ECL message control statements.

Message Control Statements. Message control statements place messages in a control log or display them on the central site operator's console. See figure 3-3 for the formats of the control statements we will discuss.

Displaying a message (@MSG). This control statement displays messages. Only the command field is required. The options C, H, I, S and W direct the message to either the communications console (C), I/O console (I), hardware console (H), or system console (S). If not specified, the S option is assumed. The W option requires a response from the S1100/60 operator. See figure 3-3 for the format.

Inserting information in the master log (@LOG). This control statement places user-specified information in the master log. Only the command field is required. The message may be up to 132 characters in length. See figure 3-3 for the format.

Exercises (210):

1. Which message control statement displays a message?

2. Which message control statement places information in the master log?

3. Write the control statement that displays the message 'PLZ RESPOND WITH THE TAPE NUMBERS OF THE IRU DUMP.' and solicits a response from the S1100/60 console operator.

211. State the function and capabilities of specified ECL symbiont directive control statements.

Symbiont directive control statements. These commands direct or redirect the contents of files to symbiont devices. The following paragraphs describe control statements related to symbiont operations. Refer to figure 3-4 as we discuss each symbiont directive control statement.

```
@[LABEL:]MSG[,Options] message

Example:  @MSG,W  Is tape reel number 01028 ready?

@[label:]LOG  message

Example:  @LOG  This is a message.
```

Figure 3–3. Message control statements.

Print output heading control (@HDG). The @HDG control statement provides a means of printing a heading or successive pages of primary output along with the print files cumulative page number. The options of the @HDG control statement provide the capability to suppress the printing of the heading, date, and page number (N). You may also start the numbering of pages with page 1 instead of the current page number (P) and/or to suppress the printing of the date and page number (X).

The example @HDG control statement in figure 3-4 causes the printing of the heading 'SBSS REPORT' at the top of each page along with the date and page number. Notice the 'P' option specifies that page numbering will begin with 1.

Symbiont file breakpointing (@BRKPT). The @BRKPT control statements provide the means of redirecting output from the generic PRINT$ and PUNCH$ files. They allow you the capability to store information for final output. There are two types of @BRKPT control statements: primary output and alternate symbiont file breakpoint. Refer to figure 3-4 as we discuss each.

Primary output breakpoint. This type of @BRKPT control statement partitions and redirects the primary output files PRINT$ and PUNCH$. Format 1 in figure 3-4 provides the format for this control statement. The generic-name parameter is either PRINT$ or PUNCH$. The part-name is the internal filename for the output to be directed to.

The 'L' option allows the stacking of multiple files as a single magnetic tape file and provides a label for each file. This can be addressed on a subsequent @SYM control statement. The label is the part-name provided by the @BRKPT control statement.

It is important to note that the part-name must be cataloged prior to using this control statement. The EXEC does an @ASG,A of the part-name. If the @ASG,A is successful, the breakpoint is performed. If the file is not cataloged, the @ASG,A cannot be successful, and the

```
@[LABEL:]HDG[,Options]  heading

Example:  @HDG,P SBSS REPORT

@[LABEL:]BRKPT[,Options] generic-name/part-name      FORMAT 1

@[LABEL:]BRKPT[,Options] generic-name                FORMAT 2

EXAMPLES:  @CAT,P  SBSS*OUTPUT.
           @ASG,A  SBSS*OUTPUT.
           @BRKPT  PRINT$/SBSS*OUTPUT
           @PRT,F  SBSS*REPORTS.
           @PRT,T  SBSS*REPORTS.
           @BRKPT  PRINT$

@[LABEL:]SYM[,Options]  filename,number,device

Example:  @SYM,U  SBSS*OUTPUT.,,NTR01P

@BK1
@BK2[,Options] [device-name[(number)]]

Example:  @BK1
          @PRT,F SBSS*REPORTS.
          @PRT,T SBSS*REPORTS.
          @BK2   NTR01P
```

Figure 3–4. Symbiont directive control statements.

breakpoint is not initiated. This is the first step. You have now partitioned a file for redirection to a symbiont device. From this point on, any output directed to PRINT$ or PUNCH$ by your run is redirected to the part-name file. To halt this process, use the alternate symbiont file breakpoint control statement.

Alternate symbiont file breakpoint. This @BRKPT control statement closes the partition of the alternate print, punch, or read files defined by the user. Format 2 of figure 3-4 provides the format of this control statement. The only option available for the @BRKPT control statement is the 'E' option. This option is used against magnetic tape files and it inhibits the end-of-file positioning for alternate read files on tape.

The example in figure 3-4 shows the usage and relationships of various ECL control statements and the @BRKPT control statements. The @CAT control statement catalogs the file SBSS*REPORTS. to the master file directory. The @BRKPT control statement partitions the PRINT$ file, assigns the SBSS*OUTPUT. file to the run, and redirects any output from PRINT$ to SBSS*OUTPUT.. The two @PRT control statements execute. The output is not displayed; it is directed and stored in SBSS*OUTPUT.. The final @BRKPT control statement closes the partitioning and sets output back to PRINT$. Notice periods are not used to punctuate filenames in the @BRKPT control statement.

You now have data stored in SBSS*OUTPUT.. You may now apply CTS or Text Editor against SBSS*OUTPUT. to manipulate or read the data, or you may output the contents to a symbiont device. Such is the use of the next control statement in our discussion, the @SYM control statement.

Symbiont output file queuing (@SYM). The @SYM control statement directs the queuing of a previously created symbiont file to a specified symbiont device for printing or punching. One of the most important options to be aware of is the 'U' option. This option inhibits the deleting of the file when processing is completed. It is important to remember this or you may inadvertently delete a file that you wished you still had. The number parameter specifies the number of copies to be printed. If omitted, one copy is assumed and the file will be printed once. A maximum of 63 copies may be specified. This device parameter specifies the symbiont device. If the device parameter is omitted, the output is redirected to the system printer.

The example @SYM control statement shown in figure 3-4 outputs one copy of the file SBSS*OUTPUT. to a device called NTR01P.

@BK1 and @BK2. This process pair permits the breakpointing and processing of files without the requirements associated with cataloging and assigning files when using the @BRKPT and @SYM control statements.

These control statements are extremely useful when processing from demand terminals, as output that would normally be sent to the terminal can be redirected (breakpointed) and input to an output file or system processor such as Text Editor.

The options available with this breakpoint processor are located on the @BK2 control statement. Options available are to call Text Editor (E) to allow you the capability to read and alter the file or to delete the file (D).

The @BK2 may also be used to specify the symbiont device and number of copies to output the file. The example in figure 3-4 compares the usage of the @BK1/@BK2 processor to the @BRKPT control statements. Both sets of statements perform the same functions.

Exercises (211):

1. What is the function of the @HDG control statement?

2. What is the function of the @BRKPT control statement?

3. What is the function of the @SYM control statement?

4. What is the function of the @BK1/@BK2 control statements?

5. What is the function of the control statement: @HDG,X RECEIPT CARD LIST?

6. What is the function of the control statement: @SYM SBSS*OUTPUT.,NTR01P,2?

212. Identify ECL dynamic runstream modification control statements.

Dynamic Runstream Modification Control Statements. These statements provide the capability to add statements or data from one runstream or file to another runstream. They also provide the capability to alter the sequence of executing control statements within a runstream. Refer to figure 3-5 for the formats and examples of the control statements as we discuss them.

Dynamic runstream expansion (@ADD). The @ADD control statement provides a means of inserting images into a runstream from any file currently assigned to the user. When the @ADD control statement is encountered, the first image of the file or element being added replaces the @ADD control statement image in the runstream. All subsequent images are taken from the file or element being added until either end-of-file or-element is encountered.

This control statement is a valuable tool for remote users (batch on demand) because control statements or data need be entered only once but may be used in many subsequent runs.

```
@[LABEL:]ADD[,Options]  filename.[element-name]

Example:  @ADD  SBSS*REPORTS.REJCLR

@[LABEL:]SETC[,Options]  f/value/j

Example:  @SETC,I 4/S4

@[LABEL:]TEST  f/value/[j[,f/value/j]]

Example:  @TEST  TE/4/S4

@[LABEL-1:]JUMP  label-2

Examples:  @JUMP  TAG
           @JUMP  4
```

Figure 3–5. Dynamic runstream modification control statements.

One of the most common options used with the @ADD control statement is the 'L' option. This option causes the control statements encountered in an added file to be printed on the demand terminal. This option is effected during demand mode only and is superfluous in the batch mode.

One common use of the @ADD control statement, as shown in the example in figure 3-5, is to add a file of reject clear card images to the runstream processing the D02 Daily Reject Listing.

Conditional control statements. Conditional control statements alter the sequence of executing control statements within a runstream. Through the use of these statements, you may set and test a control word for certain values. Based upon the test of the value, you may alter the sequence of execution by jumping to a specified label within the runstream. The conditional control statements are @SETC, @TEST, and @JUMP. Refer to figure 3-5 as we discuss each.

Condition word control (@SETC). This control statement stores a value in the control word. The control word may be accessed as a whole, halves (H1, H2), thirds (T1, T2, T3), or sixths (S1 - S6). The options A and I allow you to clear (A) or set (I) bit 30 of the condition word. The 'f' portion of the @SETC allows you to specify the type of operation to be performed such as AND, OR, and XOR. This parameter is optional. The value portion specifies the octal value (not exceeding four digits in length) to be used or entered into the designated portion of the control word. The 'j' portion of the @SETC control statement designates the portion of the condition word to be changed. If it is omitted, then T2 is assumed. Permissible entries for 'j' are T1, T2, S2, S3, or S4. Within T1 and S2, only bits 27 and 28 are modified.

The example in figure 3-5 shows that we are setting bit 30 of the control word (I) and placing an octal 4 in S4 of the control word.

Condition word testing (@TEST). The @TEST control statement tests the value of the condition word to select the particular control statements to be executed or skipped.

When more than one test is to be made, the control statement is scanned until a test is met or all the parameters are exhausted. When a test is met, the control statement immediately following the @TEST control statement is skipped.

The permissible entries for the 'f' parameter are equal (TE), not equal (NE), greater than (TG), less than or equal (TLE), even parity (TEP), or odd parity (TOP).

The value is the positive octal value (not exceeding 12 digits) to be compared with that portion of the condition word specified by 'j.' Permissible entries for 'j' are W, H1, H2, T1, through T3, and S1 through S6. If 'j' is omitted, then T2 is assumed.

Notice in the example in figure 3-5 that we are testing the S4 portion of the control word for a value equal to an octal 4. Remember if the test is satisfied, the next control statement is skipped.

Branching from within the runstream (@JUMP). The @JUMP control statement advances control to the specified labeled statement within the control stream or it may specify a decimal value (1 to 999999) indicating the number of labeled statements capable of being advanced.

We have two examples of the @JUMP control statement in figure 3-5. The first example advances control to the control statement containing the label 'TAG.' The second example advances control to the fourth control statement, capable of having a label field, following the execution of the @JUMP control statement.

The conditional runstream example in figure 3-6 really demonstrates the relationship of the various conditional control word statements. As shown, lines 1, 2, 3, 5, 7, 15, 16, 17, and 18 are processed in this sample runstream, and programs D, E, and F are executed in that order. Line 2 might be changed to set other values to produce different

```
1.  @RUN        RUNID,ACCT,PROJ
2.  @SETC       6
3.  @TEST       TE/6
4.  @XQT        A
5.  @TEST       TE/6,TE/3
6.  @JUMP       2
7.  @JUMP       X
8.  @TEST       TE/10,TE/4
9.  @JUMP       3
10. @XQT        B
11. @JUMP       Y
12. @TEST       TE/11
13. @JUMP       Z
14. @XQT        C
15. @X:XQT      D
16. @Y:XQT      E
17. @Z:XQT      F
18. @FIN
```

Figure 3–6. Conditional control examples.

runstream processing. For instance, if line 2 is changed to set the value of T2 to an octal 3, then lines 1, 2, 3, 4, 5, 7, 15, 16, 17, and 18 would be processed and programs A, D, E, and F would be executed.

Exercises (212):

1. What ECL control statement provides a means of inserting images into a runstream?

2. What control statements are categorized as conditional control statements?

3. What portion of the control word may be set by the @SETC control statement?

4. What control statement in a runstream will be skipped when all conditions of the @TEST are met?

5. What function is accomplished by the '@JUMP NEXT' control statement?

6. Write the control statements that will (a) set the S4 portion of the control word to an octal 10 and clear bit 30, (b) check the S4 portion of the control word for even parity, and (c) advance control to the fifth labeled control statement in the runstream.

3-4. Processor Control Statements

As stated in Chapter 2, system processors are programs that operate on the S1100/60 as an extension of the EXEC. The processors perform functions to create, modify, and maintain files. We will discuss two of the processors that you use within your job. These are the Data and Element Processors.

213. Cite characteristics and uses of the control statements associated with the DATA and ELT processors.

Data Processor (@DATA). The Data Processor (@DATA) creates, updates, and lists System Data Format (SDF) files from within the runstream. The '@DATA' statement calls the Data Processor. It must precede the data or correction images in the runstream. The Data Processor is terminated by a '@FIN' or '@END' statement whose SENTINEL matches the SENTINEL in the '@DATA' statement. Any statement appearing between the '@DATA' and matching '@END' statements is treated as data, except for '@FIN' and '@ADD,D' statements.

The format of the @DATA control statement is shown in figure 3-7. The options available are I, L, and U. If no option is specified, the L option is assumed. The 'L' option produces a listing of the images that were loaded or updated by the processor. The 'I' option specifies that images to be loaded to the file are immediately behind the @DATA Control Statement. The 'U' option provides you with update capability.

The parameter filename-1 specifies the file to which the data images and correction images in the runstream apply. The filename-2 parameter specifies the updated file to be generated. It is only required if you wish to leave filename-1 unaltered.

The sentinel specifies a character code of one to six characters used for comparison purposes in determining the proper terminating '@END' statement for the data mode.

Termination Sentinel (@END). The input termination sentinel (@END) marks the end of a data file or an element. It follows the data images inserted by either an '@ELT,D' or '@DATA' statement.

Notice the example in figure 3-7 shows the data processor with the I and L options. This causes the images following the @DATA Control Statement to be loaded to the SDF file SBSS*REJCLR.. The images are printed on a listing when processing is completed.

Element Processor (@ELT). The Element Processor (@ELT) introduces an element into a particular program file or makes changes to a symbolic element in a program file from within the runstream.

```
@[LABEL:]DATA[,Options]  filename-1[,filename-2[,sentinel]]

Example:  @DATA,IL  SBSS*REJCLR.
          REJCLRISU....
          REJCLRTIN....
              .
              .
              .
          @END

@[LABEL:]END  [sentinel]

@[LABEL:]ELT[,Options]  element-1[,element-2][,sentinel]

Example:  @ELT,IL  SBSS*REPORTS.REJCLR
          REJCLRISU...
          REJCLRTIN...
              .
              .
              .
          @END
```

Figure 3–7. Processor control statements.

The @ELT control statement calls the ELT processor. It must precede the element or change images in the runstream. With the exception of the @ELT,D statement, the ELT processor is terminated by the first nontransparent control statement encountered in the runstream. The @ELT,D statement is terminated by an @END statement whose sentinel matches the sentinel on the @ELT,D statement. Any control statements appearing between the @ELT,D statement and the matching @END statement are treated as data except for @FIN and @ADD,D statements. Figure 3-7 has the format of the @ELT control statement. The options A, O, R, and S identify the element type, either absolute, omnibus, relocatable, or symbolic. The 'D' option is used to specify that control statements are to be input as symbolic data. The I, L, and U options may also be used with the @ELT control statement.

As with the @DATA processor, the @ELT control statement provides the capability of identifying the input file (eltname-1) and also of identifying the file to contain the new updated images (eltname-2).

The sentinel, when the @ELT,D statement is used, is the character code which terminates the flow of data images into the element being created. The sentinel may consist of one to six characters and must agree exactly with the sentinel appearing on the @END statement used to terminate the ELT processor.

Exercises (213):

1. What control statement inserts symbolic images into a SDF file?
2. What control statement makes changes to a symbolic element in a program file?
3. What control statements may terminate the @DATA processor?
4. What is the function of the @END control statement?
5. What is the purpose of the 'I' option in the @DATA control statement?
6. What is the purpose of the 'D' option in the @ELT control statement?

3-5. File Utility Routine/Program File Utility Routine (FURPUR)

The file utility routine/program file Utility routine, also known as file utility processor or FURPUR, is an integral part of the S1100 operating system. It operates in conjunction with the executive system to provide the user with a simplified method of controlling and moving data

within the S1100 file structure. Figure 3-8 provides a summary of all 17 FURPUR commands. In this section we will discuss only the most used of the FURPUR:

(1) @CHG—change.
(2) @COPY—copy.
(3) @DELETE—delete.
(4) @PACK—pack.
(5) @PRT—list files, elements, and master file directory.

214. Associate the file utility routine/program file utility routine (FURPUR) control statements with their functions.

@CHG. The @CHG control statement is used to change the external filename, the read and write keys, and the mode (public or private). The read and write keys may also be deleted. The 'P' option changes the mode of the file from public to private. The 'Q' option sets the mode as private. The options A, O, R, and S apply to elements only.

The @CHG control statement also allows the user to change the element and version names of elements in a program file.

The example in figure 3-9 demonstrates how the @CHG control statement can be used to charge the read key of a file. Here the read key of SBSS*REPORTS is changed from 'RD–KEY' to 'READ'. Notice the old file name must be presented in its entirety. If a write key was assigned to the filename, it too would have to be identified.

@COPY. The @COPY control statement allows the

COMMAND	PURPOSE
@CHG(for elements)	Changes element name and version name of an element.
@CHG(for files)	Changes external filename, read key, write key, and mode of a file.
@CLOSE	Writes two hardware end-of-file (EOF) marks on a magnetic tape and rewinds the tape
@COPIN	Copies elements from an element file located on magnetic tape into a program file located on mass storage.
@COPOUT	Copies a program file selected elements from a program file located on mass storage onto a magnetic tape element file format.
@COPY	Copies a file or element from one file to another.
@CYCLE(for elements)	Sets the maximum number of symbolic element cycles to be retained for the specific symbolic element.
@CYCLE(for files)	Sets the maximum range of absolute F-cycles to be retained for specified files listed in the master file directory.
@DELETE(for elements)	Marks specified elements of a program file as deleted.
@DELETE(for files)	Drops cataloged files from the master file directory.
@ENABLE	Clears the disabled flag for a cataloged file.
@ERS	Erases the contents of a mass storage file. The file is still cataloged.
@FIND	Locates an element in a magnetic tape element file and positions the file immediately before the element table-of-contents block.
@MARK	Writes two hardware EOF marks on a magnetic tape file and positions the tape between the two EOF marks.
@MOVE	Moves a magnetic tape forward or backward over a specified number of EOF marks.
@PACK	Rewrites an entire program file removing specified types of elements and all elements marked as deleted.
@PCH	Punches program file elements onto 80-inch column cards.
@PREP	Creates an entry-point table from the preambles of the non-deleted relocatable elements of a program file.
@PRT	Provides a listing of the master file directory items for catalogued files, information regarding temporary files, the table-of-contents of a program file, or the text of symbolic elements.
@REWIND	Rewinds magnetic tape back to the lead point of the reel.

523213F2-95

Figure 3–8. Summary of FURPUR.

```
@CHG[,Options]  old-filname.,new-filename.

Example:  @CHG  SBSS*REPORTS/RD-KEY.,SBSS*REPORTS/READ.

@COPY[,Options]  name-1,name-2[,number]

Example:  @COPY,A  USAF*GVABSUD001.GV801F,GV801F

@DELETE[,Options]  name-1[,name-2[,...[,name-n]]]

Example:  @DELETE  SBSS*SCRATCH.

@PRT[,Options] filename.

Example:  @PRT,T  SBSS*REPORTS.

@PACK[,Options] filename.

Example:  @PACK  SBSS*REPORTS.
```

Figure 3–9. FURPUR control statements.

user to copy a file or element into another file or element. See figure 3-9 for the format and examples.

Through the use of its various options, the @COPY control statement can be proven to be quite versatile. Through @COPY, you may copy the contents of a DATA file into another DATA file (F), add a DATA file to a program file as a symbolic element (I), and through the combination of the G and M options you are able to copy sector-formatted files to tape. You may also specify the type of program file element (A, O, R, or S) to be copied. Without these options, all versions of the elements will be copied to the program file. The 'L' option is used to copy the latest cycle of a symbolic element.

The example in figure 3-9 copies the absolute element GV801F from the program file USAF*GVABSUD001. to an element called GV801F in the systems temporary file called TPF$..

@DELETE. The @DELETE control statement drops cataloged files or marks elements in program files as deleted. See figure 3-9 for format and examples.

Each cataloged file specified is marked as dropped. The filename specified may be external or internal.

When an external filename is specified, the F-cycle must be specified if it is not the latest F-cycle. If the file has read/write keys and is to be assigned to the run by the FURPUR processor, the read/write keys must be specified. The keys may be omitted if the file was assigned prior to calling the FURPUR processor. However, if a write key exists, it must be specified or the file must have been assigned to the run with the write key. If an internal filename is used, it must have been attached to an external filename by means of the @USE control statement. The file is not actually dropped until all other runs that have the file assigned to them have freed the file. When the file is dropped, the master file directory is updated.

An element may be deleted by using the A, O, R, or S options on the @DELETE control statement along with the element name. The example in figure 3-9 will delete the file SBSS*SCRATCH..

@PRT. The @PRT control statement lists files, elements, and the master file directory. A great deal of information may be obtained by the use of the various options associated with this control statement.

The 'F' option specifies information on a file from the master file directory. This information can tell the type, size, and file allocation of the file on disk. It will also tell you if the file is disabled or not. If you are trying to find reel number information, you may also find that through the use of the 'F' option.

The 'T' option produces a list of the table of contents for each specified program file or program file element. A complete rather than an abbreviated listing of the table of contents is obtained by also using the 'L' option along with the 'T' option.

A list of all files assigned to a run is obtained by the 'I' option. The 'P' option displays files for the project-id of the run.

The example in figure 3-9 produces an abbreviated list of the table of contents of the SBSS*REPORTS. file.

@PACK. The @PACK control statement removes undesired or deleted elements in the specified files and rewrites each file. The mass storage space no longer needed by the program file will be released and returned back to the system.

If no option is specified in the control statement, then all deleted elements are removed and the unused granules are released to the system. You may, through the use of the A, O, R, and S options, specify types of elements to be packed.

The example in figure 3-9 packs the program file SBSS*REPORTS.. Doing so removes all deleted elements (regardless of type) and releases their unused granules back to the system.

You should make it a rule to PACK your program files periodically. This helps manage the disk space and ensures that the space is used efficiently and effectively.

Exercises (214):

1. Match the FURPUR control statement in column B to its function in column A.

Column A	*Column B*
____ (1) Change the mode from public to private.	a. @CHG.
____ (2) Removes deleted elements from the file.	b. @COPY.
____ (3) Lists the text of a symbolic element.	c. @DELETE.
____ (4) Copies one file into another file.	d. @PRT.
____ (5) Marks elements for deletion.	e. @PACK.

2. What FURPUR control statement lists the contents of the master file directory for the file SBSS*FILE.?

3. What function is performed by the "@CHG SBSS*INLINE//KEY., SBSS*INLINE//LOCK." control statement?

4. Write the FURPUR control statement that will change the mode of the file SBSS*FILEC. to public and change the write key from 8246 to ACFG.

5. Write the FURPUR control statement that will print the complete table of contents for the program file SBSS*PROGRAMS.

Conversational Time Sharing (CTS)

THE SPERRY SERIES 1100 Conversational Time Sharing (CTS) System provides an interactive interface for users of the Series S1100 Systems.

CTS commands are conversational. CTS responds with clear error messages when commands are used incorrectly and solicits information when given insufficient information. In addition, CTS provides an information retrieval system which contains accessible information about every command in CTS. Since CTS commands are powerful, fewer commands are required to perform a task. The result is fewer mistakes and less terminal connect time.

CTS remembers what the user is doing. In particular, it remembers what program is being manipulated and what file is being used to save programs. Therefore, it is not necessary to resupply this information with each command. In similar fashion, CTS remembers other parameters about the user. These parameters may be customized for a user's special requirements.

CTS provides an automatically cataloged file system. No user file assignments are required. Users may therefore save programs between terminal sessions without the necessity of learning all about the Executive file control system.

In general, CTS is a convenient, powerful, and efficient system for problem solving, program development and debugging, and general use of the S1100 systems.

This chapter is devoted to CTS. We will cover the terms and abbreviations and symbols used in the syntax of the CTS commands. Then finally we will cover the CTS commands themselves.

4-1. Terms and Abbreviations

Before you get into a discussion of CTS, you need to understand the terms and abbreviations associated with this editor. This section will cover some of the most important terms and abbreviations of CTS.

215. Define terms and abbreviations of CTS.

Working Area - f. The working area (referred to as f) is used by CTS to save images as they are created or edited. Only data that resides in 'f' may be edited and manipulated by CTS commands. Data may be entered into 'f' and the data in 'f' may be saved for future use. The mode of 'f' may be either ASCII or Fieldata, depending on whether the data images residing in 'f' consist of ASCII or Fieldata characters. When CTS is activated, this area is automatically provided for you the user.

CTS Command. Each line entered into CTS is either a command requesting some action be taken or is a data line which is placed in the working area. A command is a line of information which does not begin with a line number, or the line number specification is ignored because it is followed by an asterisk (*). An asterisk may precede any command name but is required only when line numbers are being automatically generated and when a CTS command rather than a data image is being typed. A command begins with a command name (or its abbreviation) and usually is followed by other information (for example, line or column limits). A command normally terminates with the last nonblank character. A percent sign (%) character, followed by at least one space separates comments from the end of a command.

Data. Each line of information in the CTS working area 'f' is referred to as a data line. These data lines may be saved as a program element or as a data file. Hence, a program element or a data file may be referred to as data when it is in the working area.

Element. An element is data that has been saved (stored) in a program file. The three major types of elements are symbolic, relocatable, and absolute. A symbolic element is a series of images usually written in a program language such as BASIC. Symbolic elements are also used to store data images such as reject clear cards, receipts, etc. This is the type of element that can be edited or created by CTS. Symbolic elements are used as input by language processors, programs, and reports. Language processors create as output a relocatable element. A relocatable element is an intermediate step between a symbolic and an absolute element. One or more relocatable elements are used as input by a processor called the Collector, which creates as its output an absolute element. An absolute element is code that can be executed. SBSS programs, such as GV801A, GV801F, and GV299, are stored as absolute elements.

Program. An element (especially an absolute one) in a program file is often called a program.

Program File. A program file is a file that may contain one or more elements (programs). A program file is identified by a file name and each element is identified by an element name. An element of a program file is referenced on a CTS command by the notation element-name, file-name.element-name, or qualifier*file-name.element-name. If the file name is not specified, the save file 'F' is assumed.

Data File. In CTS, all files which are not program files are considered to be data files. Unlike program files, data files contain only one data set which is identified by the file name. The data is not identified by a separate name within the file as elements are. A data file is referenced on a CTS command by the notation file-name. or qualifier*file-name. where the period or asterisk is necessary to distinguish the data file name from an element name. A data file name has the same format as a program file name but the files are distinguished by the file contents. An attempt to reference an entire program file or an element in a data file will be diagnosed as an error. There are no restrictions on the type of data which may be placed in a data file or the usage of the file. A good example of a data file is a print file.

Save File - F. The save file 'F' is a file used to save data from one terminal session to the next. It is automatically provided by CTS. This is a cataloged file (not temporary). This file is called the save file, or 'F' for reference purposes in this chapter. Data initially resides as data lines in the working area 'f' and is put into 'F' by the SAVE command. A saved file or element can be deleted, replaced, executed, or retrieved with CTS commands. Separate save files can be created and maintained by the user.

Line Number. Each line in the working area has a line number. This line number is always a positive integer. A number must be specified when a line of data is keyed in. CTS can enter the line number automatically (see NUMBER command), in which case CTS (optionally) displays the line number before the data is entered. The line number is not part of the line, only a tag by which it is identified.

Lines may be entered into 'f' in any order, but they are considered in sequence according to ascending line numbers. Therefore, if lines 100 and 110 exist in 'f,' and a line is entered with line number 105, it will be inserted between 100 and 110. If a line is entered with a line number which already exists in 'f,' the old line is discarded and the new one takes its place.

The smallest allowable line number is 1. The largest is 262,141.

Line Pointer - p. CTS maintains a line pointer, 'p,' which is always set to a nonnegative integer. This integer is the line number of the current line. Many commands use the value of 'p' as the default specification when a line number parameter is omitted. To take advantage of this, the value of 'p' must be known. Any CTS command which references a line explicitly or implicitly will change the line pointer.

Commands for saving and restoring programs usually set 'p' to zero. When lines of data are being typed into 'f,' 'p' is set to the most recently centered line. When editing commands are used, 'p' is left equal to the line number of the last line edited.

Exercises (215):

Match each term or abbreviation in column B to its definition in column A.

Column A	*Column B*
____ (1) A nonnegative integer which designates the current line.	a. f.
____ (2) A positive integer used to identify each line in a file.	b. CTS Command.
____ (3) The file used to save programs from one terminal session to the next.	c. Data.
____ (4) File that contains one or more elements.	d. Element.
____ (5) File that contains only one data set which is identified by the file name.	e. Program.
____ (6) Each line of information in the CTS working area.	f. Program File.
____ (7) Data saved in a program file.	g. Data File.
____ (8) Element in a program file.	h. F.
____ (9) Line of information which does not begin with a line number.	i. Line Number.
____ (10) Working area used by CTS to save images as they are created or edited.	j. Line Pointer.

4-2. CTS Sign-on Procedures

CTS is a processor operating under the control of the S1100 Operating System. This processor is activated by keying in the @CTS control statement. As with any control statements, you have options to provide you with various capabilities. This section will cover the various options and actions involved with the CTS sign-on procedures.

216. Cite the elements involved with the CTS sign-on procedures.

CTS Control Statement. The format of the CTS control statement is @CTS[,options] file-name.

The control statement provides you with many options that will alter the method of sign-on and status of the working area. The options available for the @CTS control statement are F,I,N,P, and Q. The 'F' option specifies that the user wants to use a unique identifier as the filename instead of the RUN–ID. In this instance the filename chosen should be specified in the filename parameter of the control statement.

The 'I' and 'N' options affect the initialization of the CTS processor. The 'I' option forces initialization of the CTS processor. Through this option the CTS processor will be either assigned or created, whichever is needed. Remember if the filename is not specified in the control statement, the run-id is used. The processor will then create a recovery file, CTS$FILE. This file retains the contents of working area 'f' intact after the user exits for CTS. CTS then checks the system news file for any messages and prints a message reminding the user of the online assistance capability.

Since the 'I' option forces a full initialization process, it assumes that a new working area 'f' will be required and the CTS session will begin from the beginning. There is an attempt, however, to restore the CTS operating environment from the recovery file CTS$FILE. This file will exist if the previous CTS session was terminated by a system crash or if CTS has been called at least once during this terminal session. For a system crash recovery, CTS prints the message: CTS RESTART.

The 'N' option of CTS also is used to initialize CTS. It skips the checking of the news file and bypasses the printing of the online assistance message.

The 'P' and 'Q' options are used to establish the more of the working area, 'f' as either ASCII (Q) OR Fieldata (P). If neither the 'P' nor 'Q' option is used, then 'Q' is assumed.

Reentering CTS. There are two methods of reentering an established CTS session normal and after system crash.

Normal. After an exit, CTS retains the contents of the working area 'f' until an @FIN statement is encountered, so it is possible to recover CTS with the working area the same as it was just prior to the exit from CTS. This is accomplished by entering the @CTS control statement without any options and without a filename, since these were specified in the initial CTS call and are still in the recovery file.

After system crash. When a system crash occurs, the CTS recovery file is generally retained intact. CTS will print the message: CTS RESTART, in the next CTS terminal session.

Exiting from CTS - XCTS. The XCTS command causes an exit from the CTS processor. This puts the system in CONTROL MODE allowing the user to execute any valid ECL control statement. An exit from CTS using the XCTS command does not alter the working area, 'f.' The working area is retained until the terminal session is ended with a @FIN control statement. A warning message will be displayed if the working area has been edited but not saved (via the SAVE command) or replaced (via the REPLACE command) prior to exiting from CTS.

Some program errors require cancelling a current program activity. For example, an infinite loop in a program or display could tie up a terminal and processor. The @@X CIO statement can be used to accomplish this.

Exercises (216):

1. What options of the @CTS control statement affect initialization? How?

2. What is the name of the CTS recovery file?

3. What options are used to establish the mode of the working area?

4. How may a CTS session be reentered?

5. What command is used to exit CTS?

4-3. CTS Command Syntax

In the discussion of the various commands, it is important to understand some of the symbols most commonly used to represent the syntax of the CTS commands. These symbols are brackets ([,]), positive integers (i,j), keyword expressions (k), character string (s), column specifications ((c1,c2)), and line limits (L). The following is a brief description of the meaning of each of these symbols.

217. Define symbols used to represent certain syntax notations involved in CTS commands.

Brackets ([,]). Brackets identify optional parameters of the CTS command.

Positive Integers (i,j). Positive decimal integers are identified by the symbols 'i' and 'j.' These integers represent line numbers.

Keyword Expression (k). Keyword expression or character specifications are used for various purposes in certain CTS commands. For example, the 'k' parameter of the FIND command specifies =, <>, >, and < operations of the search; whereas, to the SITE command it specifies whether to print the line numbers or not.

Column Specification ((C1, C2)). These pairs of numbers specify the columns within a line for which an operation is to be performed. If either C1 or C2 is not specified, then the full column limit (1, 132) is assumed.

If C1 or C2 is a string, a search determines the column limits for each line. If a match for the string is not found in the line, no operation is performed on that line. For example, ('A', 10) would specify columns 3 through 10 (since the 10 is a number, not a string) for the line: X=A+SQR(37.8), but no operation would be done for the line: X=B+SQR(37.8).

Line Limits (L). The line limits specification 'L' defines in which working area lines are operated upon by the command. Various forms of 'L' are depicted in figure 4-1. To specify a certain data line in 'f,' you identify it as 'n,' or you may identify a series of inclusive lines by specifying "n1, n2'.' Special characters may be used to produce various results. Note that an "*" by itself affects only the line following the current line. Yet if I denote *+, then all lines following the current line are affected. The character set "*+;" could be represented by "*+10" which denotes 10 lines following the current line. As "+" denotes the current line and all following lines, the "−" is used to denote the current line and all preceding lines.

As you can see, the variations of these symbols provide enormous versatility in executing CTS commands. Try them and see what you can do with them.

n	Line number n.
A - ALL	Denotes all lines.
+0 or (space)	Denotes the current line.
*	Denotes the line following the current line.
+	Denotes the current line and all following lines.
*+	Denotes all lines following the current line.
*+i or *i	Denotes i lines following the current line.
-	Denotes the current line and all preceding lines.
*-	Denotes all lines preceding the current line.
*-i	Denotes i lines preceding the current line.

Figure 4-1. Line limits - L.

Exercises (217):

1. Match the symbol in column B to its meaning in column A.

Column A	*Column A*
____ (1) Numbers which specify the columns within a line.	a. [].
____ (2) Positive decimal integers.	b. d.
____ (3) Program name specification.	c. (c1,c2).
____ (4) Optional parameters.	d. +.
____ (5) This symbol represents a keyword or character expression.	e. n.
____ (6) This character represents a positive integer less than 262,142.	f. k.
____ (7) This symbol denotes the current line and all preceding lines.	

4-4. CTS Commands

CTS commands are divided into five categories: text editing, input/output, file management, operating information, and online assistance. To provide an easier means of discussing the commands of CTS, we will discuss them within their functional areas:

218. Relate CTS text editing commands to their functions.

Text Editing Commands. One of the most important functions of CTS is its capability to provide the user with the means of modifying files or elements. CTS provides many text editing commands. The process of editing files involves being able to set the line pointer (GO), being able to enter lines of data (MANUAL, NUMBER, and TAB), searching the file for strings of text (FIND, LOCATE), being able to actually edit data within the file (CHANGE, and DELETE) and reordering lines within 'f' (DITTO, MOVE, and RESEQUENCE). Refer to figure 4-2.

Setting the line pointer - GO. The GO command sets the line pointer 'p' to any existing line in 'f' or moves the pointer forward or backward. The first example in figure 4-2 will set the line pointer to the top of 'f.' The second example sets the pointer to line 10, so that 10 is now the current line. The third and fourth examples are used to move the pointer forward 10 lines and backward 10 lines, respectively.

Line entering commands. The NUMBER, MANUAL and TAB commands are useful in entering lines of data into 'f.'

NUMBER. The NUMBER command causes automatic line number generation. It identifies what line to start with and the increments for subsequent lines.

If an "*" is used as "i," then numbering begins at the last line generated in NUMBER mode plus the increment of 'j.' This is useful in resuming the NUMBER mode after making a correction. If an "!" is used, numbering begins at the highest line in 'f' plus the increment of 'j.' This is useful in initiating the number mode to append new data lines to 'f.'

If 'i' is not specified, 100 is assumed. If 'j' is not specified, 10 is assumed except when NUM * is used to resume numbering. In this case the last NUMBER command increment is used. NUMBER may be abbreviated to 'N.'

In the examples in figure 4-2, the first number command will cause sequential generation of line numbers 10, 20,....

n1,n2	Denotes all lines from n1 through n2 inclusive if n1 is greater than n2, the lines are referenced in reverse order.
*,n2	Denotes the line following the current line if n2 is greater than the current line number (or the line preceding the current line if n2 is less than the current line number) through n2.
n1,*	Denotes n1 through the line prceding the current line if n1 is less than the current line number or through the line following the current line if n1 is greater than the current line number.
n+i	Denotes line number n and the next i line(s) that follow (for +i) or that precede (for -i) line number n.
n+	Denotes line number n and all the lines following (for n+) or all lines preceding (for n-) line number n.
+i	Denotes the current line and i lines following the current line (for +i) or i lines preceding the current line (for -i).
!-	Denotes all lines in reverse order.
!-i	Denotes the last i lines of f in reverse order.
!,n	Denotes the lines from the end of f through line n.
!,*	Denotes all lines from the end of f through the line folowing the current line.

Figure 4–1. Line limits - L (cont'd) .

The second example will cause the sequetial numbering of lines 100, 120,....

MANUAL. This command terminates the automatic generation of line numbers caused by the NUMBER command. To enter a data line, a line number must be present. To execute a CTS command, the line number must be absent or the command must be preceded by an *. The MANUAL command is useful to terminate the automatic line numbers and allows you to enter successive CTS commands.

The example in figure 4-2 will terminate automatic line numbers generation. Without the *, the word MANUAL would be entered as data.

TAB. The TAB command defines characters to be interpreted on data input as tab characters and the columns in which the tab stops are set. Up to four tab characters may be defined on one TAB command with no more than twelve tab stops distributed among the tab characters. Each TAB command clears the tab characters defined on the previous TAB command. You may cancel any tab function by merely entering the TAB command without any parameters.

The example in figure 4-2 will establish the character ";" as the tab stop character for positions 10 and 20 in the data line while ":" is identified as the tab stop for position 15.

Search commands. The search commands search all of the parts of the working area for an occurrence of a specific character string. The FIND and LOCATE commands fall into this category. The FIND comand is much faster than the LOCATE command because it searches only fixed column positions in the line. The FIND command allows specifying a relation such as equal (=), unequal (<>),...etc. On the other hand, the LOCATE command

```
GO [L]

Example:  GO
          GO 10
          GO +10
          GO -10

NUMBER [i[,j]]

Example:  NUMBER 10
          NUMBER 100,20

MANUAL

Example:  *MANUAL

TAB  [b1 i [,i,i,...][b2 i [,i,i,...]]

Example:  TAB ; 10,20  : 15

FIND s [L] [(i)] [R[=j]] k

Examples:  FIND RPTD02
           FIND RPTD02 R
           FIND RPTD02 R=5

LOCATE s  [L] [(c1,c2)] [R[=i]]

Examples:  LOCATE RPTD02
           LOCATE RPTD02 (1,6) R
           LOCATE RPTD02 (1,6) R=5

CHANGE 's1's2' [L] (c1,c2) [R[=i]]

Examples:  ->100 ABCDEF
           ->C /XXX/ ALL
           ->100 XXXABCDEF

           ->C/1324/1234/

DELETE [L] [(c1,c2)]

Examples:  DELETE 10,20
            D 10 (20,50)
```

Figure 4-2. TEXT EDITING commands.

searches all column positions in a line. When a match occurs, the data line and line number are displayed. The FIND command may be abbreviated to 'F' and the LOCATE command to 'L.'

You may specify line limits and column limits within each command. However, since the FIND command is a fixed column search command, you need onlv to specify the beginning column to begin the search function. Both commands have two methods to search for repetitive occurrences of the character strings requested. This may be accomplished by specifying the word ALL in the 'L' parameter or by using the repeat function [R]. The 'R' by itself repeats all occurrences. By specifying "R = 5" you limit the search to the first five occurrences.

Figure 4-2 provides examples to contrast these two commands. The FIND examples show how the FIND can be used to search for the character string RPTD02 in positions 1–6 of 'f.' The first example will display only the first occurrence of the character string. The second example will display all occurrences. The third will display the first five occurrences only.

The first LOCATE command example will display the first occurrences of the character string RPTD02 it encounters. This command differs from the FIND command in that this search is unrestrictive. The character string may appear anywhere in the text. To accomplish the same function as the FIND command, column limits would have to be specified. This is demonstrated by the second LOCATE example. Here the column limits are specified as 1 through 6 and the repeat function is on. This command will perform the same as the second FIND example in that it will display all occurrences of the character string RPTD02 in positions 1 through 6 of the data line. The third example will display the first five occurrences.

The 'k' option of the FIND command is used to specify =, <>, >, and < operations.

Editing commands. The CHANGE and DELETE commands are specifically designed to help make corrections to the lines in 'f.'

Frequently, more than one of these commands will be suitable to make a particular correction. At other times, one will clearly be advantageous. When a correction is made, the corrected line, including its line number, is displayed.

The most elementary and often the easiest way to correct a line, however, is to replace it by entering a line with the same line number as the line to be changed.

CHANGE. The CHANGE command causes the string s1 to be replaced by the string s2 in all the lines of 'f' specified by 'L' but only within the column limits specified by (c1,c2). The CHANGE command may be abbreviated to C. Remember 'L' is the line specification. It may specify a range of lines within 'f' or may say ALL (or A) to identify all lines within 'f.' The CHANGE command also has the repeat function 'R' as the FIND and LOCATE commands.

The strings s1 and s2 may be different lengths and either length can be null. For example, if s1 is null, CTS will insert s2 in front of the data lines specified by 'L.' See the first example in figure 4-2.

The strings can be delimited by any nonblank character not appearing in s1 or s2. For example, /s1/s2/ is often used.

The second example of the CHANGE command will change the first occurrence of the character string "1324" to "1234."

DELETE. The DELETE command causes the lines specified by L to be deleted from 'f.' If (c1,c2) is specified and is not (1,132) or if string column limits are used, then only the specified columns of each line are deleted and the lines are packed. If the column limits are (1, 132), then the entire line and line number are removed from 'f.' If 'L' is not specified, then the current line of 'f' is deleted. Delete may be abbreviated to 'D.'

The example in figure 4-2 will delete lines 10 through 20 of 'f.' The second example will delete positions 20 through 50 of line 10 of 'f.'

Line reordering commands. The editing commands generally manipulate all or some of the characters within working area lines. The commands in this category manipulate line numbers. MOVE and RESEQUENCE assigns new line numbers to 'f' which may result in changing the order of these lines. DITTO assigns new line numbers without deleting the old ones.

DITTO. The DITTO command causes the lines of 'f' specified by 'L' to be duplicated, leaving the old lines undisturbed. The new lines are assigned line numbers starting with 'i' and in increments of 'j.' If 'j' is omitted, 10 is assumed. Figure 4-3 shows an example of the use of the DITTO command.

MOVE. The MOVE command is a destructive DITTO. In other words the lines of 'f' specified by 'L' are deleted as they are copied. The new line numbers are assigned lines starting with 'i' and in increments of 'j.' See how the two commands differ in the examples in figure 4-3. Lines 10 and 15 of 'f' are deleted by the MOVE command.

RESEQUENCE. The RESEQUENCE command causes new line numbers to be assigned to the lines of 'f' specified by 'L.' The lines are assigned numbers starting with 'i' in increments of 'j.' If 'L' is not specified, then the entire working area 'f' is resequenced. The example in figure 4-3 demonstrates how the resequence command works. Since no parameters are specified 100,10 all is assumed.

Exercises (218):

Match the function in column A to its command in column B.

Column A	*Column B*
____ (1) Copy lines 10 through 15 to lines of 'f' starting at 100, deleting the old lines.	a. GO.
____ (2) Replace the contents of XXX with ABC.	b. FIND ABCD.
____ (3) Searches for a string ABCD in positions 1–4 of 'f.'	c. CHANGE /XXX/ABC.
____ (4) Sets the pointer to the top of 'f.'	d. MOVE 10, 50 100.
____ (5) Inserts the string XXX in the first three positions of the current line.	e. LOCATE ABCD.
____ (6) Renumbers lines beginning with line 200, with lines 100, increments of 10.	f. C /XXX/.
____ (7) Duplicates lines 10 through 15 to lines 100 of 'f.'	g. DELETE.
____ (8) Prints the line of text that contains the string ABCD regardless of position in 'f.'	h. DITTO 10,15 100.
	i. RES 100,200.

```
DITTO L i[,j]

Example:  ->10 LINE 10
          ->15 LINE 15
          ->DITTO 10+ 16,5
          ->10 LINE 10
          ->15 LINE 15
          ->16 LINE 10
          ->21 LINE 15

MOVE L i[,j]

Example:  ->10 LINE 10
          ->15 LINE 15
          ->MOVE 10+ 16,!
          ->16 LINE 10
          ->21 LINE 15

RESEQUENCE [i[,j]] [L]

Example:  ->1 LINE A
          ->20 LINE B
          ->23 LINE C
          ->156 LINE D
          ->RES
          ->100 LINE A
          ->110 LINE B
          ->120 LINE C
          ->130 LINE D
```

Figure 4-3. Line reordering commands.

219. Relate input/output commands of CTS to their functions.

File Input/Output Commands. The commands in the following paragraphs transfer data between working area 'f,' a file, between two files, or delete data from a file. The commands that fall into this category are ADD, COPY, MERGE, OLD, REPLACE, SAVE, and UNSAVE. Refer to figure 4-4 as we discuss each command.

ADD. The ADD command adds the contents of the data file or program element specified by 'd' to the runstream where each statement is processed as though it was entered at the terminal. The statements may either be CTS commands or data. The example ADD command in figure 4-4 adds the contents of file SBSS*REJCLR. to 'f.'

COPY. The COPY command copies files or elements from one area to another. The d1 parameter specifies the area to be copied (input) and d2 is the area to be copied into (output). If either name is not specified, 'F' is assumed. If both the input and output element names are omitted, the entire file is copied. That is, if both files are program files, all elements are copied or if both files are data files, the data is copied. If the files are not of the same type, the COPY is not done and an error message is displayed.

You may copy a data file into a program file as an element without receiving an error message by using the "P=I" parameter. The 'P' parameter is also used to specify the type of element to copy (A,O,R and S). The example shown in figure 4-4 will copy all symbolic elements (P=S) from file FILE1. to FILE2..

MERGE. The MERGE command causes the program or data file to be merged or appended to 'f.' If 'i' is specified, the lines from 'd' are assigned line numbers starting with 'i' and of course incremented by 'j' as they are inserted into 'f.' If 'i' is null the line numbers from the input file are used. An "*" in 'i' inserts lines starting with the current line plus 'j.' An '!' for 'i' inserts lines starting with the number of the last line in 'f' plus 'j.'

The 'L' parameter identifies which lines from the input file (d) are to be read into 'f.' If 'L' is omitted, all of the input file is read.

The example shown in figure 4-4 merges lines 350, 400 from FILEA, beginning with line 200 in increments of 1.

OLD. The OLD command retrieves from 'F' an element or a data file previously saved with the SAVE or REPLACE commands. This command discards the contents of 'f' and replaces it with ALL (omit the 'L' parameter) or part (as specified by 'L') of a symbolic element or data file (d). The command changes the name of 'f' to correspond to the new contents. As with the MERGE command, the 'i' and 'j' parameters are used to change the line numbers of the data lines as they are read into 'f.' If they are omitted, then the line numbers from the input file are used. The example in figure 4-4 replaces the contents of 'f' with the contents of SBSS*REPORTS.SCHEDULE.

REPLACE. The REPLACE command causes the entire contents of the program element or data file to be discarded and the L-designated part of 'f' to be substituted in its place. The data file or element must already exist. If it does not, the SAVE command must be used. If the 'd' and 'L' parameters are omitted, the working area name assigned by the last OLD, or NEW, command is used and all of 'f' is saved. If 'f' does not have a name, CTS solicits one.

The REPLACE command is the same as an UNSAVE command followed by a SAVE command except that UNSAVE deletes all types of elements, not just symbolic, and REPLACE deletes any subroutine definition by the name 'd' in addition to the symbolic element. Since REPLACE destroys a subroutine definition, an attempt to REPLACE 'd,' where 'd' is an active CTS subroutine, should not be used.

The example shown in figure 4-4 causes the entire contents of the file or element that was specified by the last OLD or SAVE commands to be discarded and replaced by the contents of 'f.'

SAVE. The SAVE command causes the L-designated lines from 'f' to be saved as a symbolic element or data file 'F.' If 'd' already exists from a previous SAVE command, it must either be deleted first by the UNSAVE command or the REPLACE command should be used. If 'd' specifies only a filename, rather than an element or file and element, the command saves the specified lines of 'f' as the contents of a data file. The example shown in figure 4-4 saves lines 100 through 200 under the name PROG..

```
ADD d

Example:  ADD SBSS*REJCLR.

COPY[,P] [d1[,d2]]

Example:  COPY,S FILE1.,FILE2.

MERGE [i[,j]] d [L]

Example:  MERGE 200,1 FILEA. 350,400

OLD [i[,j]] d [L]

Example:  OLD  SBSS*REPORTS.SCHEDULE

REPLACE [d] [L]

Example:  REPLACE

SAVE [d] [L]

Example:  Save PROG. 100,200

UNSAVE d1[,d2[,...[,dn]]]

Example:  UNSAVE PROG.
```

Figure 4–4. Input/output commands.

UNSAVE. The UNSAVE command deletes a specified previously saved element or data file. If an element name is specified, then CTS deletes any relocatable and absolute elements in addition to the symbolic element. If a file name is not specified, 'F' is assumed. If a name is not specified, CTS solicits a name.

LIST. The LIST command prints the lines of 'f' as specified by 'L.' If 'L' is omitted, then the entire contents of 'f' are printed. If k = N the line numbers will not be listed. The example in figure 4-5 will display the entire contents of 'f' without listing the line numbers of the data lines. The column limits cause only the specified columns to be listed.

PRINT. The PRINT causes the lines of 'f' as specified by 'L' to be printed. This command is similar to the LIST command except that if 'L' is omitted, only the current line is printed. When the keyword 'k' is S, CTS prints a scale for the line to be printed. The scale is up to 72 characters long. When the keyword 'k' is N, as with the LIST command, this omits the printing of the line numbers as the data lines are displayed. This command may be abbreviated to 'P.' The example in figure 4-5 will print data lines 10 through 20 of 'f.' A scale will also be output for the lines printed.

SITE. The SITE command directs output to an onsite printer or device. The command causes those lines specified by 'L' to be printed. As with the LIST command if 'L' is omitted, then ALL is assumed and the entire contents of 'F' is output. If 'k'=N, then the line numbers of the data lines output will be omitted.

The example in figure 4-5 demonstrates how this command is executed. After inputting the command, the question 'HDG?' is displayed. Your response is printed as heading on the top of each page of the output. The next question 'RTN?' is displayed. The response indicates to whom and where the listing should be returned. The next question 'COPY?' is displayed. The response should indicate the number of copies you desire to be printed. In this case, the response is 2; therefore, the file will be output twice. The final question is 'SITE?.' The response causes the output to be printed at the specified onsite device. In this case 2 copies of the file will be printed on the onsite printer named NTR01P.

Exercises (219):

Match the function in column A to its command in column B.

Column A	*Column B*
____ (1) Substitute the entire contents of the working area 'f' with the contents of SBSS*DATA..	a. ADD SBSS*DATA.
____ (2) Delete the previously saved file SBSS*DATA..	b. COPY, I SBSS*DATA.
____ (3) Price the current line of the working area 'f.'	c. MERGE* SBSS*DATA.
____ (4) Direct output to an onsite printer on device.	d. REPLACE SBSS*DATA.
____ (5) Print the entire contents of the working area 'f.'	e. SAVE SBSS*DATA.
____ (6) Add the contents of file SBSS*DATA. to the runstream.	f. UNSAVE SBSS*DATA.
____ (7) Save the contents of the working area 'f' as the data file SBSS*DATA..	g. LIST.
____ (8) Add the contents of SBSS*DATA. to the working area 'f,' beginning with the current line plus 10.	h. PRINT.
____ (9) Copy the data file SBSS*DATA. into the working area 'f.'	i. SITE.

```
LIST [L] [(c1,c2)] [k]

Example:  LIST N

PRINT [L] [(c1,c2)] [k]

Example:  P 10,20 S

SITE [L] [(c1,c2)] [k]

Example:  ->SITE
          ->HDG?->
          ->RTN?->
          ->COPY?->2
          ->SITE?->NTR01P
```

Figure 4–5. PRINT output commmands.

220. Cite the functions and options of various file management commands of CTS.

File Management Commands. Most file management is done automatically in CTS. The working area 'f' and the save file 'F,' are automatically provided by CTS. Other files are assigned when necessary. The commands in the following paragraphs are used to create new files, delete or release files that are no longer needed, attach different names to files, and erase or compact the contents of files. Commands categorized as file management are ERASE, PACK, PURGE, RELEASE, and USE. Refer to figure 4-6 as we discuss each command.

ERASE. The ERASE command deletes all previously saved programs or data in the files specified by the commands. Notice in the format in figure 4-6 that more than one file may be specified by one command. The files specified may be either program or data files. After they are erased, they are empty and may be used again. This has the same effect as performing an UNSAVE on each program within a program file and then specifying a PACK on the file, except that the file is still recognized by the system as a program file rather than being neither type. If you do not specify any filename, CTS will ask you for the filename. Erasing and reusing a file is less costly than purging and recreating the file. The example in figure 4-6 erases the contents of file SBSS*REJCLR..

PACK. The PACK command compacts the contents of the program files as specified by the command. It eliminates the deleted (unsaved on replaced) program elements and then releases the unused space back to the system. Through the use of the A,O,R, and S options, you may specify the type of nondeleted elements to be removed from the program file. You may even specify whether you want to release (I) or not release (N) the unused granules of storage back to the system. The example in figure 4-6 removes all symbolic files from the file SBSS*REPORTS. that have been previously marked for deletion.

RELEASE. As the @FREE releases files used by the run so does the RELEASE command. This command makes the cataloged files specified by the command available for use by other users. If the files are temporary, it purges them. CTS does not assign any files for exclusive use (except for PURGE and ERASE). Many other processors, however, do require exclusive use such as FURPUR. Exclusive use cannot be obtained unless the file is released by all other runs. If you forget to specify what files you wish to release, CTS will ask you for their names. The example in figure 4-6 releases the exclusive use of file SBSS*REPORTS..

PURGE. The PURGE command decatalogs (eliminates from the master file directory) the files specified by the command. The file and its contents no longer exist. If all filenames are omitted, CTS will ask you to provide them. The example in figure 4-6 decatalogs file SBSS*REJCLR. from the master file directory.

USE. This command is used to associate an internal filename (F1) with an external filename (F2). As with the @USE control statement, the internal filename should be from 1 to 12 characters long. If the external filename has not been created, then a warning message will be printed. The example in figure 4-6 associates the internal filename 'A' to the external filename SBSS*REPORTS. From this point on within the run SBSS*REPORTS. may be refered to as 'A.'

```
ERASE F1[,F2[,...[,Fn]]]

Example:  ERASE SBSS*REJCLR.

PACK[,P] [F1[,F2[,...[,Fn]]]]

Example:  PACK,S  SBSS*REPORTS.

PURGE F1[,F2[,...[,Fn]]]

Example:  PURGE SBSS*REJCLR.

RELEASE F1[,F2[,...[,Fn]]]

Example:  RELEASE  SBSS*REPORTS.

USE  F1,F2

Example:  USE A,SBSS*REPORTS.
```

Figure 4-6. File management commands.

Exercises (220):

1. What command will remove the exclusive use of files to the CTS run?

2. What option of the PACK command will cause the granules of storage not to be released back to the system?

3. What is the purpose of the USE command?

4. What status will the file be in after an ERASE command is performed against it?

5. Write the CTS command that will make the cataloged file SBSS*REPORTS. available for use by other users.

6. Write the CTS command that will eliminate the deleted relocatable elements of the file USAF*PROGRAMS. and release the unused granules back to the system.

```
LIST CATALOG

LIST FILE [F1[,F2[,...[Fn]]]]

Example:  LIST FILE

LIST INUSE

LIST SAVED[,P[,i]] [d1[,d2[,...[dn]]]]

Example:  LIST SAVED,D

DATE

LENGTH

      IMAGES = 1505  LAST LINE NUMBER = 40000

MAIL  runid

LOOK

HELP
```

Figure 4-7. Operating information and online assistance commands.

221. Cite the functions and options of the CTS commands which provide operating information and online assistance.

Operating Information. Operating information commands provide you with information concerning filenames, assignments, size, and contents. Also available is information concerning the DATE, length of working area, and news. Follow figure 4-7 as we discuss each command in the following paragraphs.

LIST CATALOG. The LIST CATALOG command lists the files which were cataloged under the user's project-id even if they have not been previously referenced or assigned by this run. This command finds the filenames by searching the master file directory. This is a time-consuming and costly function; therefore, you should use this command with discretion.

LIST FILE. The LIST FILE command lists the file information (size, date, etc.) for the files specified by the command. If no file names are specified, (as shown in the example in fig. 4-7), the user's assumed program file, 'F,' is used.

LIST INUSE. The LIST INUSE command lists the names of all cataloged and temporary files currently assigned to the run. Information such as external filenames, internal filenames attached through the use of the USE command, equipment type, and the assigned options of each file are displayed for each file in the form:
Qual*File,equipment,assign-options use names.

LIST SAVED. The LIST SAVED command lists information contained in the table of contents of the files or elements specified by the command. The command lists the elements saved last, first. All the parameters shown in the format are optional. If none are specified, then the assumed name last given to 'F' will be used. The 'P' option of the command provides you with many options. Of course there are the standard A,O,R and S options which are used to limit the listing of particular types of elements with in the program file. You may list deleted (unsaved or replaced) elements as well as nondeleted elements from the program file by the use of the 'D' option. This option identifies the deleted elements by preceding its name by an asterisk (*). If you wish to check the date and time that an element was created and the size of the element in addition to its type and name, use the 'L' option. The 'V' option is used to list only those elements in the program file that have the same version name as specified in each of the element names on the command. Note that even though you must specify the element name, it will be ignored and the version name will be used. If you do not use any options, then the command defaults to the A,O,R, and S options.

The 'i' parameter is used to specify how many element names you wish to list. If 'i'=5 then the last 5 elements from each specified profile file are listed. The example in figure 4-7 lists the deleted and nondeleted elements of 'F.'

DATE. The DATE command prints the system date and time. This may be useful in a subroutine or to time stamp an output listing.

LENGTH. The LENGTH command prints the number of lines of data in the working area 'f,' and the line number of the last line in 'f.' In many instances, printing the number of lines of data in 'f' is helpful in determining if the

previous OLD, MERGE, DITTO, etc., command has done what was expected. Figure 4-7 shows how the LENGTH command prints the message indicating the total number of lines and the last line number in 'f.'

MAIL and LOOK. The MAIL command allows you to establish a message file of up to 10 lines in length for a specified period. If you don't specify the run-id, CTS will ask you for it. After the command is executed, CTS solicits 10 lines of input. If your message is less than 10 lines, enter STOP to terminate the input. The message file is opened by the receiving user by the use of the LOOK command. The LOOK command searches to see if any mail has been sent to the user's run-id. If mail is found, the messages, the run which sent it, and the date it was sent are displayed. If mail is not found or is found but cannot be read, the message: YOU HAVE NO MAIL . . . is displayed.

Online Assistance. The HELP command activates the HELP processor of CTS which provides an explanation of CTS commands, the syntax of the commands, or the meaning of various fields in commands. When the HELP command is entered, CTS responds with: WHEN YOU NO LONGER NEED HELP, TYPE EXIT TEACH? - TYPE YES OR TYPE A HELP COMMAND.>

A YES answer causes a description of the 10 HELP subcommands (shown in fig. 4-8) to be printed. After this description or instead of the YES response, one of the HELP subcommands can be entered. Each of the HELP subcommands displays the requested information except EXIT and ask for another HELP command with the question: MORE HELP?>.

This sequence continues until this question is answered with an EXIT command or a NO which returns control back to CTS.

This function is extremely helpful from the most inexperienced user to the expert. It provides a list of all CTS commands, grouped by function (COMMANDS), for when you need to jar your memory about which command you desire to use. If you forget the meaning of certain terms such as working area, save area, etc., the DEFINITION subcommand will provide you with the definitions of the specified terms.

The syntax, fields, and description of the CTS command are provided by the SYNTAX, FIELD, and EXPLAIN subcommands respectively.

Try to become familiar with the HELP function of CTS. It will help you smooth out some rough spots of your CTS processing and may provide you further insights into CTS processing.

Exercises (221):

1. What command provides a list of all cataloged and temporary files currently assigned to the run?

EXIT	Return control to CTS.
EXPLAIN [CTS-command]	Prints a brief description of the specified CTS command.
SYNTAX [CTS-command]	Prints the exact syntax of the specified CTS command.
FIELD[i] [CTS-command]	Prints the meaning of the i-th field of the specified CTS command.
LENGTH specification	Prints the definition of the line limit specification.
TEACH	Prints a description of the HELP commands.
USE	Prints an explanation of some typical CTS commands.
COMMANDS	Lists all CTS commands, grouped by function.
SYMBOLS	Explains the meaning of the symbols used by HELP to define the CTS commands.
DEFINITIONS	Explains the meaning of terms, such as working area.

Figure 4–8. Help subcommands.

2. In what sequence does the LIST SAVED command list the elements of a program file?

3. What option of the LIST SAVED command will list all deleted and run-deleted elements of a program file?

4. What is the purpose of the LENGTH command?

5. What command establishes a message file of up to 10 lines in length for a specified run-id?

6. What command activates the online assistance function of CTS?

7. Write the CTS command that will list the symbolic elements of file SBSS*REPORTS. that have been marked for deletion.

CHAPTER 5

Text Editor (ED)

IN THE PREVIOUS chapter we discussed Conversational Time Sharing. As an editor it has many advantages. There is, however, another editor available on the S1100/60 which you should be familiar with called Text Editor. This chapter will be concerned with the Text Editor (ED) processor.

The ED processor is a text editor which allows the user to create or edit a symbolic file or element conversationally. Insertion, deletion, and replacement of text are allowed. Both ASCII and fieldata text can be edited.

5-1. @ED Control Statement

To invoke the Text Editor and specify its input, output, and operational modes, you need to become familiar with the @ED control statement. This section deals with identifying the various options and parameters associated with Text Editor and its operational modes.

222. Cite the function and options of the @ED control statement and identify the operational modes of the Text Editor processor.

Control Statement. To call the Text Editor processor and specify its input, output, and operational modes, you need to process the @ED control statement. The format for the @ED control statement is @ED,OPTIONS name-1.[,name-2.] The options available are shown in figure 5-1. The name-1 specifies an input file or element. The name-2 specifies an output file or element. Here is an example of the @ED control statement and an explanation of it: @ED,I SBSS*REPORTS.D02. This control statement identifies that this is the initial insertion of symbolic data. The Text Editor will automatically enter the input mode. Data may now be entered into the element D02 of the file SBSS*REPORTS..

Modes. The ED processor operates in three modes; input, edit and read only. In the input mode all lines entered are directly inserted into the text. This mode is used to initially build your files and runstreams. The edit mode is used to update an existing file or element. Various commands may be used to modify your text. To change from one mode to the other, you input a blank line. The ED processor allows any editing command to be submitted while in input mode. The format is: @EDIT command, where *command* denotes any command which is valid in edit mode. The letters EDIT must be in columns 2 through 5, and the command must begin in column 7. If the command is omitted, the ED processor will return to edit mode. The read only mode lets you read the file only. You may not change nor input to the file.

Options. As you can see in figure 5-1, there are many options available to you. We will discuss some of the most common ones.

The 'I' option is used for the initial insertion of data. You use this option to create new files or elements. The editor processor is placed in the input mode and the file is ready to receive data.

The 'N' option is used to suppress printing of changed, found, or relocated lines. This provides for faster processing; however, since the changed lines are not printed, this option should be used with care.

The 'P' and 'Q' options are the same as in CTS. They specify the mode of the files. The 'P' option specifies fieldata and the 'Q' option specifies ASCII data.

The 'R' option is for read-only. You may print the file or element but no changes can be made. If changes are necessary, execute the UP command to change to the update (Edit) mode.

The 'U' option specifies that you want to update the file or element. You use this option to update your runstream files, for instance, to change a report select card from a daily option to a weekly option.

Exercises (222):

1. What is the purpose of the @ED control statement?

2. What is the purpose of the input mode?

3. What is the purpose of the edit mode?

4. How do you change from the input mode to the edit mode?

Option Character	name-1	name-2	Description
Not I, R, or U	Input	Output	Input is taken from name-1 and the resultant text is placed in name-2.
Not I, R, or U	Input	None	R option assumed for symbolic element. U option assumed for data file.
A	Input	Output	Attempt auto recovery
C	Input or Output	Ignored	Enter input mode (I option) if element does not exist. Otherwise, assume U option
D	Input	Output	Demand mode when using a batch terminal - output listing of the ED processor run will contain solicitation messages.
I	Output	Ignored	Initial insertion of symbolic input from the runstream which causes the ED processor to enter the input mode. The images following the @ED control statement are inserted into the file or element name in name-1. The I option takes precedence over the R or U option. If the I option is specified or if two fields are specified, then on entry to the text editor a check is made on the output file (i.e., field 1 if option; filed 2 if two fields are specified). If the output is a data file and the user specified an element in that file, than the ED processor error terminates after printing the error: FILE NOT PROGRAM FILE FORMAT If the output file is a program file and the user did not specify an element in that file, then the ED processor prints a warning message: WARNING: ON EXIT OUTPUT FILE WILL BECOME A DATA FILE
L	Input	Output	Print all lines following the @ED control statement. The lines printed are indented and preceded by four asterisks.
N	Input	Output	Suppresses printing of changed, found, or relocated lines.
P	Input	Output	Output file will be fieldata.
Q	Input	Ouptut	Output file will be ASCII.
R	Input	Ignored	Input is taken from name-1 of the @ED control statement; no output text is produced (read-only mode). The up command should be used if the user wishes to apply changes.
U	Input (Output if no name-2)	Output	Update the symbolic element by applying corrections and create a new symbolic element cycle. For SDF files the original images will be replaced with the updated ones. In particular, an output element name may be specified which need not be the same as the input element; the output element will have the same cycle information as would the updated input element if there were no second element specified. The T option letter is reserved for internal use to facilitate this feature. In the case that the current cycle is 62 and the number of cycles in existence is less than the maximum permitted for the element being updated, the output cycle number will be the smaller of n and m-1, where n is the number of cycles in existence and m is the maximum number of cycles permitted.
Y	Input	Output	If the output file is assigned to another run, the user is restricted to READ ONLY mode.

GT505408E

Figure 5-1. Text editor options.

```
ADD  name num1 num2

Example:  ADD PROG1. 200 300

APPEND

CHANGE /s1/s2/n G

Example:  CHANGE /D02/D20/ALL

DELETE num1 num2

Examples:  DELETE 10
           DELETE 10 20

DITTO num1 num2

Example:  DITTO 10 50

INSERT

MOVE num1 num2

Example:  MOVE 10 50

RETYPE text

Example:  RETYPE This information will replace the current line.
```

Figure 5-2. Edit commands.

5. What option on the @ED control statement specifies the output file will be ASCII?

6. What option of the @ED control statement specifies a read-only mode?

5-2. Text Editor Commands

In the previous section we discussed the different parameters and options of the Text Editor. To make that information more meaningful, you should understand what to do once you initiate the Text Editor. In this section we cover the commands of the Text Editor.

223. Specify the function of certain Text Editor edit commands.

Edit Commands. Edit commands allow you to alter the work space. Utilizing these commands, you may add, change, or delete lines of data within the file or move, insert, or undo previous actions. The various edit commands are ADD, APPEND, CHANGE, DELETE, DITTO, INSERT, MOVE, and RETYPE. Refer to figure 5-2 as we discuss each command.

ADD. The ADD command adds all or portions of a file to the current file. The example in figure 5-2 adds the contents of the file PROG., lines 200 through 300, to the current file.

APPEND. The APPEND command allows you to go to the end of the element or file and enter the input mode. This command is very useful if you have been in the edit mode correcting text and now wish to start inputing new text at the end of the file. This command may be abbreviated as 'A.'

CHANGE. The CHANGE command makes corrections in the text by replacing string-1 with string-2. The CHANGE command may be abbreviated as 'C.' This command searches a specified number of lines for string-1. When and if string-1 is found, string-2 is substituted for it. The 'n' parameter specifies the number of lines to be scanned. If you want to scan the whole file, then enter the word ALL. The 'G' parameter specifies the number of occurrences of string-1. The example in figure 5-2 changes all occurrences of the string D02 in the current file to D20.

DELETE. The DELETE command deletes lines from

the text. You can delete one specific line or a group of lines. The first example in figure 5-2 deletes line 10 of the current file. The other example deletes lines 10 through 20 of the current file.

DITTO. The DITTO command duplicates lines of the text. The duplicated lines are inserted immediately after the present cursor position. At the completion of the DITTO, the pointer is positioned at the last line inserted. The example in figure 5-2 duplicates lines 10 through 50 of the current file and places them after the current line. Note that lines 10 through 50 will still remain in the file.

INSERT. The INSERT command allows the user to insert a line of text following the current line. This command can be abbreviated as 'I.'

MOVE. The MOVE command directs the editor to move lines of text from their original position so that they follow the current line. Unlike the DITTO command, the MOVE command deletes the text at the original position. The example in figure 5-2 moves the data from lines 10 through 50 of the current file and places them behind the current line. Note that lines 10 through 50 will be deleted.

RETYPE. The RETYPE command completely replaces the current line with the new line following the first blank after the command. RETYPE may be abbreviated as 'R.' An example of the RETYPE command is given in figure 5-2.

Exercises (223):

1. What command is used to add a portion of a file to the current file?

2. What command allows you to go to the end of the file and enter the input mode?

3. What is the function of the CHANGE command?

4. What command is used to delete one specific line from the text?

5. What command duplicates lines of text without destroying the original lines?

6. What command allows the user to insert a line of text following the current line?

7. What command duplicates lines of text and destroys the original lines?

8. What command is used to completely replace the current line with a new line?

9. Write the Text Editor command that will scan five lines changing all occurrences of 'XZY' to 'XYZ.'

10. Write the Text Editor command that will delete lines 25 through 45 of the file.

11. Write the Text Editor command that will duplicate lines 75 through 100 and not delete the old lines.

224. State the function of certain Text Editor search commands.

Search Commands. The search commands allow you to search the entire text for key phrases or characters. The commands within this category are FC, FIND, LC and LOCATE. Refer to figure 5-3 as we discuss each command in the following paragraphs.

FC. The FC command searches the text for an image which corresponds exactly, column for column, starting at column one with the mask in the command. All occurrences of the image in the rest of the text are printed. The mask is used to identify the exact location of the string in relation to its position in the text. By specifying '*n' or ',n' you may limit the number of occurrences or lines searched by the command. The examples in figure 5-3 show how these options may be used with this command. The first command lists all occurrences of the string 'D02' in positions 1 through 3 of the text. The next example lists only the first five occurrences of the string, while the last example lists all occurrences of the string 'D02' in positions 1 through 3 for the next five lines within the text.

FIND. The FIND command is similar to the FC command except it lists the first occurrence only. FIND may be abbreviated as 'F.' The example in figure 5-3 demonstrates the variations of the FIND command. The first example lists the first occurrence of the string D02 in positions 1 through 3 of the text. The next example searches the next 5 lines but lists only the first occurrence of the string. The last example will search the current line only for the occurrence of the string 'D02' in position 1 through 3 of the text. The command 'FIND?' prints the current default value of the FIND command.

```
FC mask
FC*n mask     *n - stops the search after n occurrences
FC,n mask     ,n - limits the search to n lines

Example:  FC D02
          FC*5 D02
          FC,5 D02

FIND mask
FIND,n mask   ,n - same as FC
FIND. mask    '.'  - search the current line only

Examples:  FIND D02
           FIND,5 D02
           FIND. D02

LC s
LC*n s         *n - same as FC
LC,n s         ,n - same as FC

Example:  LC D02
          LC*5 D02
          LC,5 D02

LOCATE s
LOCATE,n s    ,n - same as FC
LOCATE. s     '.'  - same as FIND

Example:  LOCATE D02
          LOCATE,5 D02
          LOCATE. D02
```

Figure 5–3. Search commands.

LC. The LC command is similar to the FC command in that it searches the text for specific occurrences of an image. But where the command FC uses a mask, the LC command does not. The LC command searches for a given string of characters and lists all occurrences of the string in the remaining text. It is not position oriented. Figure 5-3 gives examples that point out the variations of this command. The first example lists all the lines of text that contain the string 'D02', no matter where the string 'D02' is located within the text. The next example lists the first five occurrences, as the last example searches the current line only.

LOCATE. The LOCATE command searches the text for a given string of characters. The search begins at the line following the current line (as with the other search commands) and proceeds sequentially through the text until a find is made or the end-of-file is encountered. This command may be abbreviated as 'L.' The example in figure 5-3 demonstrates the variations of this command. The first example lists the first occurrence of the string 'D02' no matter when it appears in the text. The next command limits the search to the first five lines of text, while the last command searches only the current line of text. By entering LOCATE? you may display the current default value of the LOCATE command.

Exercises (224):

1. What is the function of the FC command?

2. What is the function of the FIND command?

3. What is the function of the LC command?

4. What is the function of the LOCATE command?

5. Write the Text Editor command that will print the first five occurrences of the string 'ABCD' in positions 1 through 4 of the file.

6. Write the Text Editor command that will search the current line only for the string 'RPTD02' regardless of where it appears in the line.

225. Identify the function of various Text Editor Control commands.

Control Commands. Control commands are used to exercise control over the text editor processor. You may set or reestablish currency, exit the processor with corrections, or exit the processor without corrections. You may also establish automatic recovery of the work area. Refer to figure 5-4 as we discuss each command in the following paragraphs.

AUTO. The first control command we will discuss is the AUTO command. This command specifies that an automatic save of the current file is to be performed. This protects your work against any processor or system failure that may occur.

You specify in the format parameter field the frequency the auto save will be performed. If you specify 10, then after 10 inputs alter the file, the auto save will be performed. This command saves valuable time if data is lost due to a system crash or through any other abnormal termination of your run. Entering auto without any parameter causes an immediate save and turns the auto save function on.

```
AUTO n
AUTO*

GI n

Example:  GI 10

number

Example:  +20
          -20

SET tab1 tab2 ... tabn

Example:  SET 10 20 30

TAB  char

Example:  TAB ;
```

Figure 5-4. Control commands.

EXIT. The EXIT command is used to take a normal exit from the editor processor. The format is EXIT in positions 1–4. All corrections will be applied to the designated file and a normal exit will be taken.

GI. The GI command positions the editor at a line numbered according to the input text, even after the input file has been edited with insertions, deletions, and other modifications. The example in figure 5-4 positions the editor to line 10 of the text.

If the desired line has been deleted or altered in some way, the message "Requested line deleted or altered" is printed, and the editor is positioned at the next existing line. This command is useful when a user is editing text while working from a listing. This command cannot be used with print files.

LAST. The LAST command directs the editor to move to the last line in the file and stay in the edit mode. The format is LAST in positions 1–4.

The last line cannot be printed or altered after this command until the file position (line number) indicated as the last line is reentered. There are four commands which are illegal after the LAST command has been used until a line has been added or the file position is changed. These commands are CHANGE, DELETE, INLINE, and RETYPE.

NUMBER. The number command positions the editor at a desired line in the text. The 'number' is any line number you wish to go to. If you wish to go to line number 10, then input 10 and transmit.

When the specified line is located, it is typed out modifications may be made to it. If you desire to insert lines before line 1, then 0 may be typed in. This will position the editor immediately prior to the first line. This command differs form the GI command in that it uses the current line number, not the old, as is the case of the GI command. The examples will cause the editor to move forward (+20) or backward (−20) lines of text in the file.

INPUT. The INPUT command directs the editor to enter the input mode. The format is INPUT in positions 1–5.

In this mode everything which is typed in is inserted in the file until an exit from the mode is taken. This is especially useful when large volumes of input are to be entered. Exiting from this mode is accomplished by entering a blank line.

OMIT. The OMIT command is used if you have made changes to the file and then decide you do not want the corrections to be applied to the file. The format is OMIT in positions 1–4. When this command is executed, all changes made to the file are ignored and you exit text editor.

SET. The SET command sets the tabs for the commands which allow them (insert, retype, find, FC, and input mode).

TAB. The TAB command specifies what character is to be used as a tabulation character. The format is TAB, one space, and the character desired. The example in figure 5-4 establishes the semicolon as the TAB character.

This character is recognized on the INSERT, RETYPE, FIND, and FC strings and on all input when in the input mode. The character is not transmitted to the file and behaves just as a tab on a typewriter.

Entering the TAB command with no character specified disables the tab character.

UNDO. The UNDO command causes all changes made since the last return to the top of the file to be ignored and positions the editor at the top of the file. The format is UNDO in positions 1–4.

UP. The UP command causes the editor to change from the read-only mode to the update mode. The format is UP in positions 1 and 2.

Exercises (225):

1. What is the function of the AUTO command?

2. What control command will exit the editor processor and apply all corrections to the file?

3. What is the function of the GI command?

4. What control command directs the editor to move to the last line in the file and stay in the edit mode?

5. What is the function of the number command?

6. What control command directs the editor to enter the input mode?

7. What is the function of the OMIT command?

8. What control command is used to set tabs?

9. Write the Text Editor command that specifies the exclamation point (!) as the tabulation character.

226. Identify the function of various Text Editor input/output and miscellaneous commands to their function.

Input/Output Commands. The input/output commands are used to send messages to the console operator, display files on the VDU, print a file on the onsite printer, or to punch a file on an onsite punch. The commands within this category are OPR, PRINT, and SITE. Refer to figure 5-5 as we discuss each command in the following paragraphs.

OPR. The OPR command sends a message to the console operator. You can either send a message informing the operator of a current condition or situation, or you can send a message requesting a response from the operator. These two methods are identified by the examples given in figure 5-5.

PRINT. The PRINT command prints specified lines of text on the VDU screen. You can specify a series of lines such as in the first example in figure 5-5. This command displays the contents of lines 10 through 20 on the VDU screen. The next example displays the next 10 lines. The third example displays the entire file from the top to the last line. The last example displays the remaining file starting with the line after the current line.

SITE. The SITE command outputs lines of text to an onsite on remote device, printer, or punch. It is very similar to the SITE command of CTS except the device is specified in the command itself and not as a solicited response. The SITE command provides you the capabilities to output selected portions of the file (format 1), specify how may lines to output (format 2), or to output the entire file (format 3).

The example in figure 5-5 outputs the entire file to the device NTR01P. You could have requested a heading for the output and a message also if you wished.

Miscellaneous Commands. The commands that fit into this category are MSCHAR, SCALE, IB, and NEXT. Refer to figure 5-5 as we discuss the commands in the following paragraphs.

MSCHAR. The MSCHAR command sets a character which is to be translated as the master space symbol '@' when it is input in column one of input lines in the input mode. The example in figure 5-5 translates a '?' as the master space character. To disable this capability, simply input this command without any special character.

SCALE. The SCALE command causes a line to be printed which can be used as a measuring scale for column sensitive operations. The standard scale is positions 1 through 72. You may alter the beginning of the scale or set a range for the scale.

This command is useful when making report select cards for a file or element. Many times you need to use options in the select card and the scale lets you see the exact columns. The first example in figure 5-5 produces a full scale, the next example begins the scale at position 10, and the last example establishes a range of positions 10 through 50 for the scale.

IB. The IB command is used to insert text before the current line. You would use this command to insert a blank line into the text. The blank line would go on the line above the current line.

NEXT. The NEXT command moves the line pointer to the next line of text. If you were at line 20 and input the next command, then line 21 would be displayed and available for editing. The format is NEXT in positions 1–4.

```
OPR  message
OPR* message

Examples:  OPR Mount tape reel number 01005, PLZ.
           OPR* Mount BLANK tape.  RESPOND with REEL NBR.

PRINT n1 n2
PRINT n
PRINT !
PRINT +

Examples:  PRINT 10 20
           PRINT 10
           PRINT !
           PRINT +

SITE n1 n2 device
SITE n device
SITE device

Example:  SITE NTR01P
          HDG?->
          MSG?->

MSCHAR char

Example:  MSCHAR ?

SCALE
SCALE n1
SCALE n1 n2

Examples:  SCALE
           SCALE 10
           SCALE 10 50

IB
```

Figure 5–5. Input/output and miscellaneous commands.

Exercises (226):

1. What command is used to send a message to the console operator and request a response?

2. What input/output command is used to display lines of text on the VDU screen?

3. What command is used to direct output to an onsite printer.

4. What command translates the master space character as an ']'?

5. What command is used to insert text before the current line.

6. What command moves the line pointer to the next line?

7. Write the Text Editor command that will allow you to insert a line before the current line.

8. Write the Text Editor command to produce a scale for positions 25 through 70 of the current line of the file.

CHAPTER 6

SBSS Software

AS WE HAVE seen in the previous chapters the S1100/60 system uses many software tools to perform its job. Yet this is merely for the S1100/60 system itself. For each ADS to perform its function, it needs its own software. The SBSS software works with the S1100/60 to accomplish its tasks. This chapter is devoted to the discussion of the SBSS software and how they interact with the system software of the S1100/60.

6-1. Types of Software

In this section, we will discuss the types and uses of the software used in the SBSS.

227. Cite the types and uses of software used in the SBSS/ADS.

AFM 67-1, Vol 2, Part Four, identifies 3 major types of SBSS programs (System Support, Application, and Utility). In this section I would like to add a fourth type to our discussion of SBSS programs. This fourth type is called Drivers.

System Support Programs. These programs establish and provide the operating environment for the SBSS/ADS. These programs preposition the required constant/variable data in memory (initialization); provide input/output capability via drivers for peripherals and communication subsystems, contain common subroutines such ad DMS/CALC, locate details, decimal to binary/binary to decimal conversion routines, and allow for an orderly restore if records updated by transactions subsequently reject.

Application Programs. All programs loaded to the program bank, except for system support and utility, are designated as application programs. Application programs are called into memory to process specific inputs, produce reports, or other off-line products. Application programs interface with system support programs to perform input/output functions and access the common functions.

Utility Programs. These are self-contained programs, working independently of the system support programs. The program numbers GV001-GV099 correspond to the category of utility programs. Most utility programs are self-adjusting and function as directed by the parameters provided by the user.

Drivers. These programs are used to control the processing of the applicable software. Drivers are responsible for selecting appropriate runstreams from the master ECL file, modifying it for the appropriate options, and starting the runstream. Programs associated with drivers are GV208A, GV208B, GV801B, GV898A/B/C/D, GV956A, GV956/J-N/P/R. The driver concept will be discussed fully later in this chapter.

Exercises (227):

1. List the four types of SBSS programs.

2. What type of SBSS program controls the processing of applicable software?

3. What programs are self-contained programs, working independently of the system support programs?

6-2. SBSS Programs

This section discusses the program number assignment of SBSS programs and some of the files associated with SBSS programs.

228. Cite the concepts and principles of SBSS programs.

Standard system code/ADS identifier GV is assigned to the SBSS. The GV prefix controls SBSS-related documentation and software during the distribution phases. Since the implementation, enhancements have required some variation in the internal handling of unique application routines. SBSS programs applicable to the batch processing mode (Twilight/Reports) appear in the program bank index (PBI) with a prefix of GV. A program compiled/mapped for online application (inline) using High Volume Transaction Interface Processing (HVTIP) will appear on the PBI with a 'D' prefix; however, all output will reflect GV for distribution purposes.

The standard S1100/60 (SBSS/GV) program identification numbers assigned for SBSS applications consists of five alphanumeric characters and are assigned to the categories and subcategories as shown in figure 6-1.

PROGRAM ID NUMBER	CATEGORY
GV000	UNUSED
GV001-GV050	General purpose and software application utility programs which are primarily used for 1100/60 ADPE/PCAM support operations.
GV051-GV099	Specialized SBSS application utility programs which are primarily used for direct support of USAF Supply operations.
GV100-GV199	Special purpose and one-time application utility programs that are used for one-time SBSS file conversion, maintenance, or other infrequent applications.
GV200-GV299	Applications support and supervisory programs.
GV300-GV399	Recurring specialized and utility programs for DSDO/LGS/ACL inhouse applications
	GV300-GV319 USAF Management Data Bank (LGSMA)
	GV320-GV334 Quality Control/Product Review (LGSFP/LGSFE)
	GV335-GV349 File Maintenance (LGSDS)
	GV350-GV369 Item Accounting/Inventory Management (LGSDI)
	GV370-GV389 Supply Reports (LGSDR)
	GV390-GV399 Accounting and Finance/Materiel Systems (ACLA)
GV400-GV599	File Maintenance
GV600-GV799	Item Accounting
GV800-GV919	Reports
GV920-GV999	Accounting and Finance

523213CF3-96

Figure 6–1. SBSS programs categories/subcategories.

Exercises (228):

1. What is the significance of the GV prefix of the program number?

2. How many characters makeup the SBSS/GV program number?

6-3. SBSS Filename Conventions

Now that you have an understanding of the different types of programs associated with the SBSS, let's look at how the filenames for SBSS programs are designed. This section discusses the makeup of the three different types of filenames used within the SBSS.

229. Identify the elements of the filenaming conventions for SBSS files.

General Filename Format. The filename for all SBSS-created files follows this general format: GVxxxxCMyyy. GVxxxx is the 5–6 character program number that created the file. The 'C' parameter identifies the security classification of the file. The codes are 'S' for secret, 'C' for confidential, and 'U' for unclassified. The 'M' parameter identifies the media code, 'T' for tape or 'D' for disk. The three position number 'yyy' identifies the number of files crated by this run of the program. For example, if the report D02, Daily Reject Listing, created a file on disk, it would be named: GV818UD001.. GV818 is the program identification number for the report D02. The 'U' identifies this file as unclassified and its mode is that of a disk file (D). You can also tell that this is the first created by this run of GV818 (001).

Qualifiers. There will always be a qualifier assigned to the filename. There are three different types used in the SBSS.

The first qualifier we will discuss is the one that is assigned to all print/punch output files. The qualifier is generated by program GV801A and is comprised of the following components: nBPSrrrzzz. The 'n' parameter specifies the gang number (1–8). The 'BPS' is a constant identifier. The number represented by 'rrr' is a number from the master batch production schedule (BPS) file and is the sequence number of GV801A executions. The number represented by 'zzz' is the relative number of selects that is processed by this execution of GV801A. As an example,

the print file created by the D02 could be 8BPS001001*. The '8' identifies that the file was created by a run processed on the secondary gang. The first '001' number identifies the file as the first file created by the D02. The second '001' identifies the run as the first select executed within this run of GV801A.

Now look back at the last two examples and you will see a complete filename for a file created by the D02: 8BPS001001*GV818UD001.. This is how you identify the output files created by SBSS programs. Understanding the filename conventions will be imperative to performing your job well.

The second type of qualifier identifies files created by one run that is to be used by another run and the SELECTS are ALWAYS submitted within the same execution of GV801A. The format is nBPSrrr. As with the previous qualifier the 'n' parameter identifies the gang number that the run was processed on. The 'BPS' is a constant identifier. The number identified by 'rrr' is the sequence number of executions of program GV801A. An example of this type qualifier is 1BPS003. It was created on gang 1 by the third execution within GV801A.

The third type of qualifier used is when one runstream creates a file that is used as input by another, but their SELECTS may be processed by different executions of GV801A. The format is nUSAF. Again the 'n' identifies the gang number. The USAF is constant. For example, program UTL030 creates a file on the master file directory (MFD) which identifies the tape numbers of tapes used. The file name for this file is DUMP030. If the file was generated on gang 8, the qualifier would be 8USAF. The filename then would be 8USAF*DUMP030.. Understanding these simple rules will enable you to be able to clearly identify the type and function of the different files created by SBSS programs.

Exercises (229):

Answer the following questions by referencing this filename: 8BPS002001*GV818UD001.

1. What is the security classification of the file?

2. What program created the file?

3. What gang was the file produced on?

4. Is the file a disk file or a tape file?

6-4. SBSS Files

This section is devoted to the discussion of various files which are used in the processing and control the SBSS/ADS. They are not all the files used in the SBSS but some of the more important.

230. Cite the purpose and relationships of files and elements used in the SBSS processing environment and the programs that employ them.

USAF*GVECLUD001.(Master ECL File). This file contains the tables and runstreams used by the SBSS drivers to select the proper application program and ECL runstream to process SBSS transactions. Some of the elements within this file are discussed below.

TRICS table. This table is contained as an element on the file USAF*GVECLUD001. It contains all the transaction identifier codes (TRIC) of the inline transactions (fig. 6-2). It is used by program GV801A (batch driver) to identify inline transactions. It also provides GV801A the following information:

Modes (inline, twilight, etc.) the transactions can process in.

Whether or not a transaction changes the mode (example: inline to twilight).

Which transactions require special editing.

The element name of the runstream in USAF*GVECLUD001. that processes the transaction.

Whether or not the transaction can process on the 'Primary' or 'Secondary' Data Base (Concurrent Processing).

The account-id for 'RUN' statements (first statement in table).

SELECTS table. This table contains the identifier codes of the twilight, reports, and utility transactions (SELECTS). It is used by GV801A to identify the transactions that process in the three modes mentioned. Basically, it provides the same information for SELECTS as the TRICS table provides for inline transactions. See figure 6-3 for an example of the SELECTS table. See figure 6-4 for a detailed explanation of the fields within both tables.

ECL runstreams. USAF*GVECLUD001 also contains the runstreams necessary to process all SBSS transactions. They are identified by the names of the absolute program with an 'R' suffix. For example, the runstream element to execute program GV210 is GV210R. There is a runstream element on the ECL file for each SBSS SELECT transaction. Therefore, the same program may be executed by more than one element. See figure 6-5 for an example of element GV210R contained in USAF*GVECLUD001.

nBPsrrr*GVECLUD001[Local ECL files]. This file contains the runstream created by GV801A used to process SBSS transactions. The 'n' identifies the gang and the 'rrr' identifies the restart number for that particular execution of GV801A. See figure 6-6 for an example of the runstream for the execution of GV210R after modification by GV801A.

```
USAF
A01      3INYYYN000GV221BR1
A02      3INYYYN000GV221BR1
A04      3INYYYN000GV221BR1
A05      3INYYYN000GV221BR1
A0A      3INYYYN000GV221BR1
A0B      3INYYYN000GV221BR1
A0D      3INYYYN000GV221BR1
A0E      3INYYYN000GV221BR1
A21      3INYYYN000GV221BR1
A22      3INYYYN000GV221BR1
A23      3INYYYN000GV221BR1
A24      3INYYYN000GV221BR1
A25      3INYYYN000GV221BR1
A2A      3INYYYN000GV221BR1
A2B      3INYYYN000GV221BR1
A2C      3INYYYN000GV221BR1
A2D      3INYYYN000GV221BR1
A2E      3INYYYN000GV221BR1
A41      3INYYYN000GV221BR1
A42      3INYYYN000GV221BR1
A43      3INYYYN000GV221BR1
A44      3INYYYN000GV221BR1
A45      3INYYYN000GV221BR1
A4A      3INYYYN000GV221BR1
A4B      3INYYYN000GV221BR1
A4C      3INYYYN000GV221BR1
A4D      3INYYYN000GV221BR1
A4E      3INYYYN000GV221BR1
AB1      3INYYYN000GV221BR1
AB2      3INYYYN000GV221BR1
AC1      3INYYYN000GV221BR1
AC2      3INYYYN000GV221BR1
AC3      3INYYYN000GV221BR1
AC4      3INYYYN000GV221BR1
AC5      3INYYYN000GV221BR1
ADJ      3INYYYN000GV221BR1
AE1      3INYYYN000GV221BR1
AE2      3INYYYN000GV221BR1
AE3      3INYYYN000GV221BR1
AE4      3INYYYN000GV221BR1
AE5      3INYYYN000GV221BR1
AE9      3INYYYN000GV221BR1
AF1      3INYYYN000GV221BR1
AF2      3INYYYN000GV221BR1
AF3      3INYYYN000GV221BR1
AF4      3INYYYN000GV221BR1
AF5      3INYYYN000GV221BR1
AFC      3INYYYN000GV221BR1
AFX      3INYYYN000GV221BR1
AFZ      3INYYYNOOOGV221BR1

                                GT5059D8E
```

Figure 6–2. USAF*GVECLUD001.TRICS.

```
RPTD02    6BY   YYY005GV818R        0200NY
RPTD36    6BY    YN006GV802R        3602NY
1SSD04    6BN    YN000GV804R        0436NY
1SSD05    6BY    YN000GV856R        0504NY
1SSD14    6BN    YN000GV807R        1405NY
1SSD15    6BY    YN000GV829R        1514NY
1SSD20    6BN    YN000GV821R        2015NY
1SSD21    6BN    YN000GV852R        2120NY
1SSD23    6BN    YN000GV854R        2321NY
1SSD24    6BN    YN000GV825R        2423NY
1SSD25    6BY    YN000GV841R        2524NY
1SSD28    6BN    YN000GV868R        2825NY
RPTD06    6BN    YN014GV800R        0628NY
RPTD13    6BN    YN015GV833R        1306NY
RPTD07    6BN    YN008GV980R        0713NY
RPTD11    6BN    YN000GV969R        1107NY
RPTD12    6BN    YN000GV982R        1211NY
RPTD01    6BN    YN000GV992R        0112NY
RPTD10    6BN    YN000GV981R        1001NY
RPTD29    6BN    YN000GV958R        2910NY
RPTD27    6BN    YN000GV993R        2729NY
RPTD16    6BN    YN000GV883R        1627NY
RPTD26    6BY    YN000GV899R        2616NY
RPTD09    6BY    YN000GV885R        0926NY
RPTD22    6BN    YN000GV985R        2209NY
RPTD32    6BN    YN000GV949R        3222NY
RPTD30    6BN    YN000GV897R        3032NY
RPTD17    6BN    YN000GV990R        1730NY
RPTD33    6BY    YN000GV907R        3317NY
RPTD31    6BN    YN000GV983R        3133NY
RPTD08    6BN    YN021GV975R        0831NY
RPTD18    6BN    YN022GV820R        1808YY
RPTD19    6BN    YN023GV846R        1918YN
1XTD19    6BN   Y N000GV846RD       0000YN
RPTD03    6BN    YN024GV834R        0319NY
RPTM01    6BN    YN000GV972R        0000NY
RPTM02    6BY    YN000GV879R        0000NY
RPTM03    6BN    YN000GV991R        0000YY
RPTM04    6BN    YN000GV850R        0000YN
1XTM04    6BN   Y N000GV850RM       0000YN
RPTM05    6BN    YN000GV995R        0000NY
RPTM06    6BN    YN000GV903R        0000NY
RPTM07    6BN    YN000GV987R        0000NY
RPTM08    6BY    YN000GV998R        0000YY
RPTM09    6BN    YN000GV834RM       0000NY
RPTM10    6BN    YN000GV836R        0000YN
RPTM11    6BY    YN000GV832R        0000NY
RPTM12    6BN    YN000GV982RM       0000NY
RPTM13    6BN    YN018GV855R        0000NY
RPTM14    6BY    YN000GV814R        0000YY
RPTM15    6BY    YN000GV844R        0000YYY
1XTM15    6BY   Y N000GV844RM       0000YY
RPTM16    6BY    YN000GV845R        0000YY
1XTM16    6BY   Y N000GV845RM       0000YY
1XTXYZ4179BY    Y N000GV417R        0000YY
1XTXXX4189BY    Y N000GV418R        0000YY

                                GT5059D8E
```

Figure 6–3. USAF*GVECLUD001.SELECTS.

Field	Position	Explanation
Select ID	1-9	This is the ID of the first card ready by bath program. ALL SELECT-IDs must be in the table.
ID Site	10	This field contains the number of bytes in the SELECT-ID.
I-B Code	11	This field identifies the SELECT-ID as a batch (B) or inline (I) transaction.
Data Code	12	This field contrains a 'Y' when data cards follow the program select card. Otherwise, it will contain an 'N'.
Mode Flags	13-15	Position 13 represents inline mode. Position 14 represents twilight mode. Position 15 represents reports mode. When one of the above fields contain a 'Y', the program select card can process in the mode represented. Otherwise the field will contain an 'N' or blank.
Chg Mode	16	If field contains'Y', the program that processes the program select card will change the process mode (EXAMPLE: inline to twilight). Otherwise the field will contain an 'N'.
Edit Code	17-19	This field contains a relative number that is used by GV801A. The number directs GV801A to the paragraph that edits the program select card. If the field contains zeroes, the select is not edited by GV801A.
RUN ID	20-31	This field contains the element name of the runstream on USAF*GVECLUD001 that will process the program select card.
520 - OCCUR-RANCE	32-35	Flags in the Reports Sequence Control Record (record code 520) are set when the mandatory dailies are executed. Positions 32-33 contain the relative number of the flag associated with the program select card. Positions 34-35 contain the relative number of the program select card that must process first. If the field contains zeroes, the program select cards are not mandatory dailies.
Dual System Code	36-37	If position 36 contains a 'Y', the select can be processed on the Primary Data Base (concurrent processing) System. If position 37 contains a 'Y', the select can be processed on the Secondary Data Base (concurrent processing) System. If either field contains an 'N', GV801A will reject the program select card when executes win the represented system.
Blank or Remarks	38-80	This field may be blank or contain pertinent remarks.

NOTE: The only dual system (primary or secondary) edits performed by GV801A are those defined under the explanation of Dual-System Code. Any other edits must be performed by the applicable application programs.

GT50590RE

Figure 6–4. SELECT/TRIC breakdown.

```
1   ?RUN        RRRRRR,AAAAAAAAAAAA,PPPPPPPPPPPP,9999,9999/9999
2   ?SETC,I     .
3   ?FREE         TPF$.         .
4   ?ASG,T        TPF$..F///5ØØØ     .
5   ?ASG,A        USAF*GVABSUDØØ1.   .
6   ?COPY,A       USAF*GVABSUDØØ1.GV898A,TPF$.    .
7   ?COPY,A       USAF*GVABSUDØØ1.GV9Ø1B,TPF$.    .
8   ?COPY,A       USAF*GVABSUDØØ1.GV8Ø1D,TPF$.    .
9   ? . COPY,A       USAF*GVABSUDØØ1.GV211C,TPF$.    .
10   ?COPY,A       USAF*GVABSUDØØ1.GV211B,TPF$.    .
11   ?COPY,A       USAF*GVABSUDØØ1.GV21Ø,TPF$.    .
12   ?FREE         USAF*GVABSUDØØ1.   .
13   ?XQT     TPF$.GV898A .
14   1
15   DATA2
16   DATA3
17   ?TEST       TE/Ø/S2   .
18   ?JUMP       SKP8Ø1    .
19   ?XQT     TPF$.GV21Ø   .
20   DATA1
21   ?TEST       TNE/Ø/S2        .
22   ?TEST       TE/Ø/T2   .
23   ?JUMP       SKP8Ø1    .
24   ?XQT     TPF$.GV211B .
25   ?TEST       TE/Ø/S2   .
26   ?JUMP       SKP8Ø1    .
27   ? . XQT     TPF$.GV211C .
28   ? . TEST       TE/Ø/S2   .
29   ? . JUMP       SKP8Ø1    .
30   ?XQT     TPF$.GV8Ø1B .
31   DATA2
32   ?TEST       TE/Ø/S2   .
33   ?JUMP       SKP8Ø1    .
34   ?XQT     TPF$.GV898A .
35   2
36   DATA2
37   DATA3
38   ?TEST       TNE/Ø/S2        .
39   ?JUMP       SKP81D    .
40   ?SKP8Ø1:   .
41   ?PMD,PALE        .
42   ?XQT     TPF$.GV8Ø1D .
42   ?JUMP     DOPRT   .
44   ?SKP81D;    .
44   ?SYM,D     PRINT$
44   ?DOPRT;
45   ?END
```

GT505908E

Figure 6–5. USAF*GVECLUD001.GV210R.

```
1 @RUN       GV001,USAF,1BPS500001,9999,9999/9999
2 @MSG,N  PRINT DEFAULTS TO HOS
3 @QUAL 1BPS500
4 @SETC,I    .
5 @FREE        TPF$.       .
6 @ASG,T       TPF$.,F///5000    .
7 @ASG,A       USAF*GVABSUD001.   .
8 @COPY,A      USAF*GVABSUD001.GV898A,TPF$.    .
9 @COPY,A      USAF*GVABSUD001.GV801B,TPF$.    .
10 @COPY,A      USAF*GVABSUD001.GV801D,TPF$.    .
11 @ . COPY,A      USAF*GVABSUD001.GV211C,TPF$.    .
12 @COPY,A      USAF*GVABSUD001.GV211B,TPF$.    .
13 @COPY,A      USAF*GVABSUD001.GV210,TPF$.    .
14 @FREE        USAF*GVABSUD001.   .
15 @XQT    TPF$.GV898A .
16 1
17 QUAL=1BTS500      KEY=1GV001                    GV210R
18 INTL
19 @TEST      TE/0/S2   .
20 @JUMP      SKP801    .
21 @XQT    TPF$.GV210   .
22 INTL
23 @TEST      TNE/0/S2       .
24 @TEST      TE/0/T2   .
25 @JUMP      SKP801    .
26 @XQT    TPF$.GV211B .
27 @TEST      TE/0/S2   .
28 @JUMP      SKP801    .
29 @ . XQT    TPF$.GV211C .
30 @ . TEST      TE/0/S2   .
31 @ . JUMP      SKP801    .
32 @XQT    TPF$.GV801B .
33 QUAL=1BPS500      KEY=1GV001                    GV210R
34 @TEST      TE/0/S2   .
35 @JUMP      SKP801    .
36 @XQT    TPF$.GV898A .
37 2
38 QUAL=1BPS500      KEY=1GV001                    GV210R
39 INTL
40 @TEST      TNE/0/S2       .
41 @JUMP      SKP81D    .
42 @SKP801:   .
43 @PMD,PALE      .
44 @XQT    TPF$.GV801D .
45 @JUMP    DOPRT   .
46 @SKP81D:    .
47 @SYM,D    PRINT$
48 @DOPRT:
49 @END
```

GT5089088E

Figure 6–6. nBPSrr*GVECLUD001.GV210R.

Batch Production Scheduler (BPS) Files. The BPS files are indexed sequential files. These files cannot be read nor altered by Text Editor and CTS processors. They are built and can only be accessed by the program GV801/GV898 series (which will be discussed fully in the next section). The purpose of BPS files is to control runs in the SBSS ADS batch environment. There are two BPS files: The MASTER BPS file (USAF*GV801CUD001.) and local BPS files (nBPSrrrGV801AUD001.). The master BPS file provides the batch driver complex the prerequisite information of the batch runstream. This file is created and maintained by program GV801C. The local PBS file is created by GV801A. There is one master record for each program/runstream in batch (local BPS file). There is one master record for each program/report select card (master BPS file). These files also contain prerequisite records which follow the master record for runstreams which must run after another runstream has been completed. These files ensure runstreams and programs are processed in the correct sequence when entered in the same batch runstream.

Program Bank Index Files. The SBSS absolute programs used for SBSS processing are maintained in sequential files. The following is a list of the files.

USAF*GVABSUD001.
SBSS*DBALIB$.
DMS*DBALIB$.
TIP$*TIPABS$.
GVCMP*ABS.

This list is but a sample of files needed to maintain and support the SBSS/ADS. You as an analyst should become knowledgeable of these files. More information on these files may be found in AFM 67-1, Vol 2, Part Four, Chapter 13.

Exercises (230):

1. What is the purpose of the master ECL file?

2. What element is used by GV801A to identify those transactions that are processed in reports mode?

3. What does the local ECL file contain?

4. What is the purpose of the master batch production schedule file?

5. List the files used in the Program Bank Index.

6-5. Driver Complex

In the previous chapters and sections we have discussed the Executive System, ECL, CTS, Text Editor and SBSS Software. Now is the time to put them all together and discuss the interrelationships of the system and application software used to process SBSS transactions and run units.

231. Cite the functions of the programs and subroutines used within the driver complex.

Programs and Subroutines. Before we get into a heavy discussion of the Driver Complex, we need to cover the different programs and subroutines used within the Driver Complex. The following are some of the major contributors to the Driver Complex.

GV205A—system standard subroutines. This subprogram assigns temporary application support files and stores SBSS audit records, transaction histories, document images, and punch images. It is called by the SBSS application programs.

GV205B—standard/common SBSS subroutines. This is a special library program which contains subroutines to accomplish square root functions, computes Julian date difference, builds the MILSTRIP requisition number, updates requisition number, and other common functions.

GV208A—SBSS functional system analyzer (FSA). Program GV208A processes all inline transactions entered by the Pseudo Reader, GV208B, Program GV324, and any TIP/TIWADS device. The program determines which inline program must process the input transaction. GV208A passes this information and the transaction to GV209A for processing.

GV208B—SBSS requirements driver. GV208B drives the releveling and followup processes by creating the appropriate LVL or FUP inline transaction and passing it to GV208A processing and then submits to GV208A. This continues until releveling and followup is complete.

GV209A/B—inline driver/batch driver. GV209A/B serves as the communication link between the programs required to process the SBSS transactions. They perform the program calls.

GV210—SBSS initialization routine. This routine provides the RPS/main site operator the capability to initialize the SBSS system for either inline or offline processing.

GV211A/B—end-of-transaction online/batch. This program is responsible for performing the following functions for all inline and batch programs.

a. Transfer of output data (documents, REJ/MGT notices, alternate print/punch files) generated by the inline or batch process to the output function or area designated to receive the output data.

b. Assignment of transaction serial numbers and the writing of transaction history records to the SBSS data base record area (DBRA).

c. Logging of audit trail data generated by the inline batch and Program GV211A/B to the system log file.

d. Storing of punch images on the SBSS data base.

GV214—inline/twilight symbiont processor. Program

GV211 calls GV214 to output documents and alternate print/punch file to a specified batch device.

GV215—reject and restore. This program provides the SBSS application programs with a standard method of processing reject and error conditions. Each reject message consists of the input image, reject phrase, date, time, override flags, and up to four lines of supplemental data lines. A suspense record is written to the daily reject suspense record. A restore condition restores the data base to its original condition. Each reject may be sent to additional terminals including the terminal originating the transaction. Each of the above conditions may be overridden and any combination may occur.

GV221B—pseudo reader. GV221B interfaces with the SBSS batch driver (GV801B) for the loading and processing of inline and twilight transactions submitted through the RPS/main card reader. The transactions are placed in one of three queues and then passed to the inline or twilight driver for processing. The frequency of processing transactions from the pseudo queue can vary from 1 to 3 inputs.

GV801A—batch production scheduler. GV801A preprocesses all SBSS transactions entered via a batch or demand device, inserts the select cards into the appropriate ECL runstreams, and starts program GV801B which starts the runstreams created by GV801A.

GV801B—reports/batch driver. GV801B performs the @START of the ECL runstreams created by GV801A.

GV801C—production file update. GV801C creates the master batch production scheduler file (BPS).

GV801D—reset active batch production schedule gang flag. GV801D provides error recovery for abnormal termination of runstreams. GV801D flags the local BPS record of the error and removes the active flag for the runstream. This allows the next execution of program GV801A. Otherwise, the batch driver system assumes that a program GV801A run is active and rejects all other executions.

GV801E—batch production schedule file cleanup. GV801E executes in runstream GV898CR which processes the select card RPTRUN. It deletes the local ECL (nBPSrrr*GVECLUD001) and BPS (nBPSrrr*GV801AUD001) files that were created by program GV801A during twilight and reports mode processing.

GV898A—update handler. GV898A is included in all batch program runstreams to compute processing time and formats end-of-job messages. This program also updates the report processing data record, reports sequence control record, and the batch production schedule file. After processing is completed, GV898A calls program GV898B to output the end-of-job message.

GV898B—message handler. GV898B is called by batch application programs to send management notices and reject notices to the RPS console VDU/printer.

When the calling program sends a valid reject code, Program GV898B reads the reject code's reject notices record from the data base to obtain a plain language description of the reject and displays the information on the RPS console. If the calling program sends zeros for the reject code, Program GV898B assumes the program provided its own message in the message file and displays it on the RPS console. After program GV898B displays the message in either of the above situations, it accepts input from the RPS terminal. This provides application programs the ability to request input from the RPS terminal.

GV898C—report end-of-night (RPTEON). GV898C reads and edits the reports sequence control record to ensure that all daily and monthly reports have been processed. This program reads the special control record and updates the input count record for each terminal. It also resets the necessary flags on the special control record so next-day processing can begin.

GV898D—RPS message receiver. GV898D acts only as a receiver for a response asked the RPS console operator (RPS MSC) and passes the response to the application program.

GV956A—A&F inline driver. GV956A serves as the communication link between the programs required to process the SBSS transactions. GV956A performs the program calls.

Exercises (231):

1. What program is the SBSS functional system analyzer?

2. What is the function of GV214?

3. What program performs the @START of the ECL runstreams created by GV801A?

4. What is the function of GV898D?

5. What is the purpose of GV956A?

6. What program drives the releveling and followup process?

7. What program transfers output data generated by the inline process to the output function?

8. What program provides SBSS programs with a standard method of processing reject conditions?

9. What program creates the master batch production scheduler file?

10. What program provides error recovery for abnormal termination of runstreams?

11. What program deletes the local ECL and BPS files?

12. What program edits the reports sequence control record to ensure daily reports have processed?

232. Cite the basic concepts and principles of the driver complex.

Now that you have the purpose and functions of most of the programs and subroutines used within the Driver Complex, we can continue our study of the Driver Complex.

Driver Complex. The Driver Complex may be divided into two categories: online and batch. The online driver controls the processing of SBSS inline transactions while the batch drivers handle SBSS batch processing (twilight reports, utilities, etc).

Online driver complex. Refer to figure 6-7 and follow it as we cover the online Driver Complex.

Inline transactions are usually submitted through a TIP/TIWADS terminal, pseudo reader, and LVL/FLP transactions. The TIP/TIWADS programs control all input and output from and to TIP registered terminals. When a terminal is registered with TIP, it is also identified to an ADS (*Example:* SBSS/GV) which has a functional system analyzer (FSA) program. Any transaction or data keyed to the terminal when signed on to TIP is passed to the ADS and its FSA program for editing.

GV208A is the FSA program for SBSS. It is executed by TIP/TIWADS when data is transmitted to TIP/TIWADS via a SBSS registered terminal or passed by an executing program (*Example:* GV221B, GV208B). GV208A receives and edits the data. If the data is a response from the RPS operator to a RPS message, GV208A writes the messages to RPS 898B RESPONSE and instructs TIP/TIWADS to print the reject message on the terminal that transmitted the erroneous data.

The subroutines GV209A and GV956A are the same. GV956A processes the Accounting and Finance transactions and GV209A processes the others.

Now let's look at figure 6-8 for a summary of how the online driver works.

During transaction processing, subroutines GV209A/GV956A, GV205A, GV205E, and GV205D remain in core. All other subroutines are only placed in core when called by the driver subroutine (GV209A/GV956A). When an online subroutine is in core, it can call GV205A, GV205D, or GV205E directly but cannot call another online subroutine.

Subroutine GV209A is the online driver subroutine for SBSS programs; GV956A is the online driver subroutine for A&F programs. Both subroutines are identical. Because they are the driver subroutines, when TIP/TIWADS executes (per GV208A's request) their associated run unit, they execute first. GV209A/GV956A performs the following functions:

a. Reads the transaction (TRIC), input PID, and the first subroutine to call.

b. Initializes the common storage area GVCSCOMM.

c. Calls GV205B to assign file SBSSCRATCH01 and to log the input image.

d. Calls the first subroutine.

When the first subroutine completes its functions and writes the next online subroutine to call in COMMON STORAGE, it returns control to GV209A/GV956A.

GV209A/GV956A reads from the COMMON STORAGE the next online subroutine and calls it. This continues until all subroutines within the TRIC process are complete. GV209A/GV956 then calls GV211 to process file SBSSCRATCH01 and terminates when GV211 returns control.

If the TRIC process is terminated because of a reject, GV209A/GV956A cancels all data base updates via a DMS-1100 rollback option. Otherwise, GV209A/GV956A commits the updates.

The non-exec online subroutine reads and updates the SBSS data base. Records of the update are stored in a work area. The non-exec online subroutine is responsible for formatting and storing the DD Form 1348-1 documents, punch images, and transaction histories created in the work area. When the non-exec online subroutine finishes its processing, it returns control to the online driver (GV209A/GV956A) to call the next non-exec online subroutine required to complete the transaction.

To delay outputs until a transaction (TRIC) successfully processes, non-exec online subroutines call GV205A to write their DD Form 1348-1 documents, log images, punch images, and transaction history transactions to disk file SBSSCRATCH01. COMMON STORAGE is the media used by the non-exec online subroutines and GV205A to transfer the data. GV205A does not call any programs but is always the recipient. After GV205A writes the data to the work file, it returns control to the calling program. Non-exec online subroutines call GV205D to store data on or read data from a work file (205 ARG) because of limited space in COMMON STORAGE.

Online programs request the online driver (GV209A/GV956A) to call GV215 when a fatal error condition exists. GV215 obtains the reject information left by the online program in COMMON STORAGE. GV215 obtains the reject phrase definition of standard reject error codes from the reject notices record and calls GV205A to write the reject data to SBSSSCRATCH01.

The online driver subroutine (GV209A/GV956A) calls GV211A when the processing of the transaction (TRIC) is completed—the driver has satisfied all requests to call

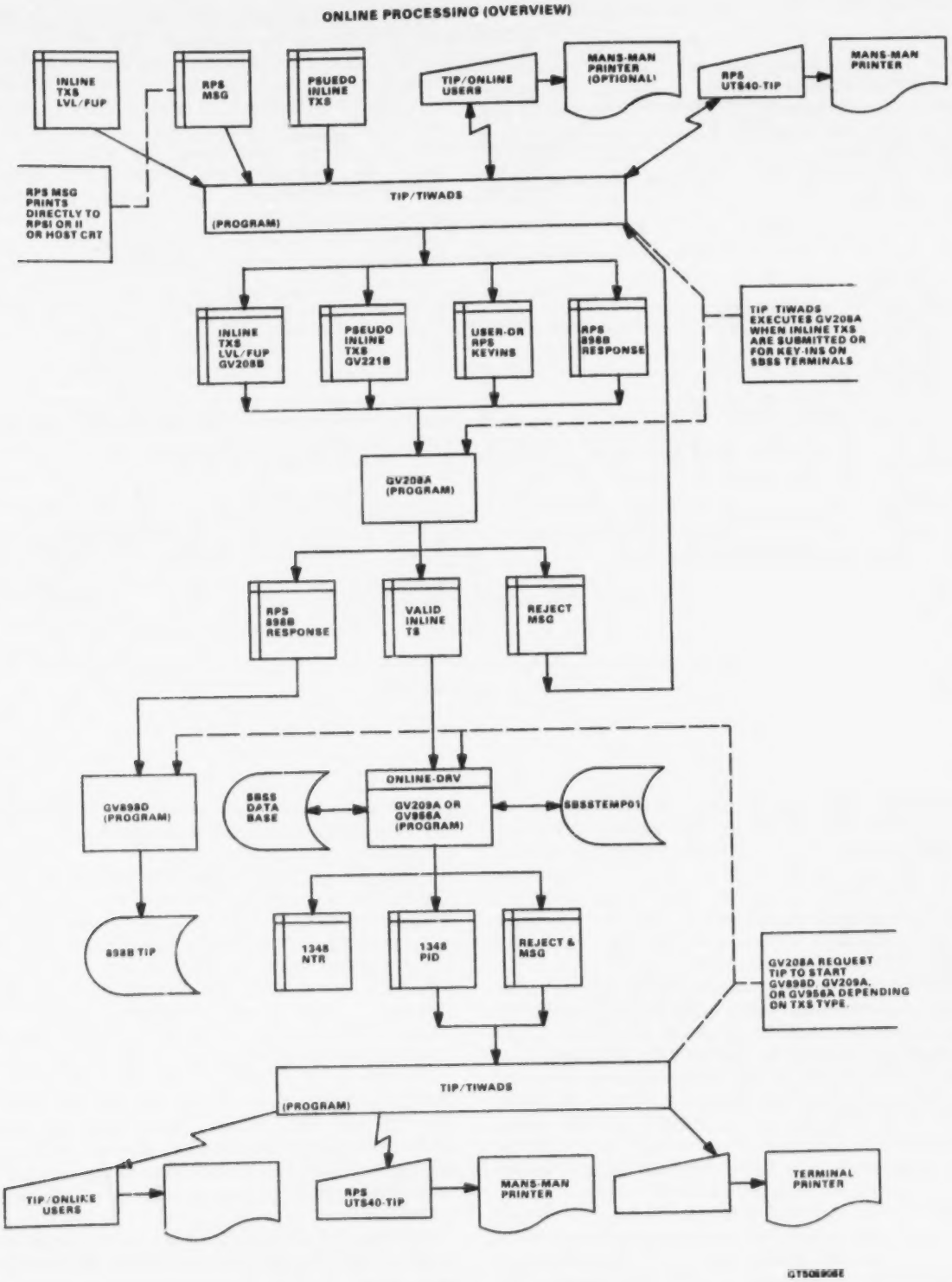

Figure 6-7. Online driver complex.

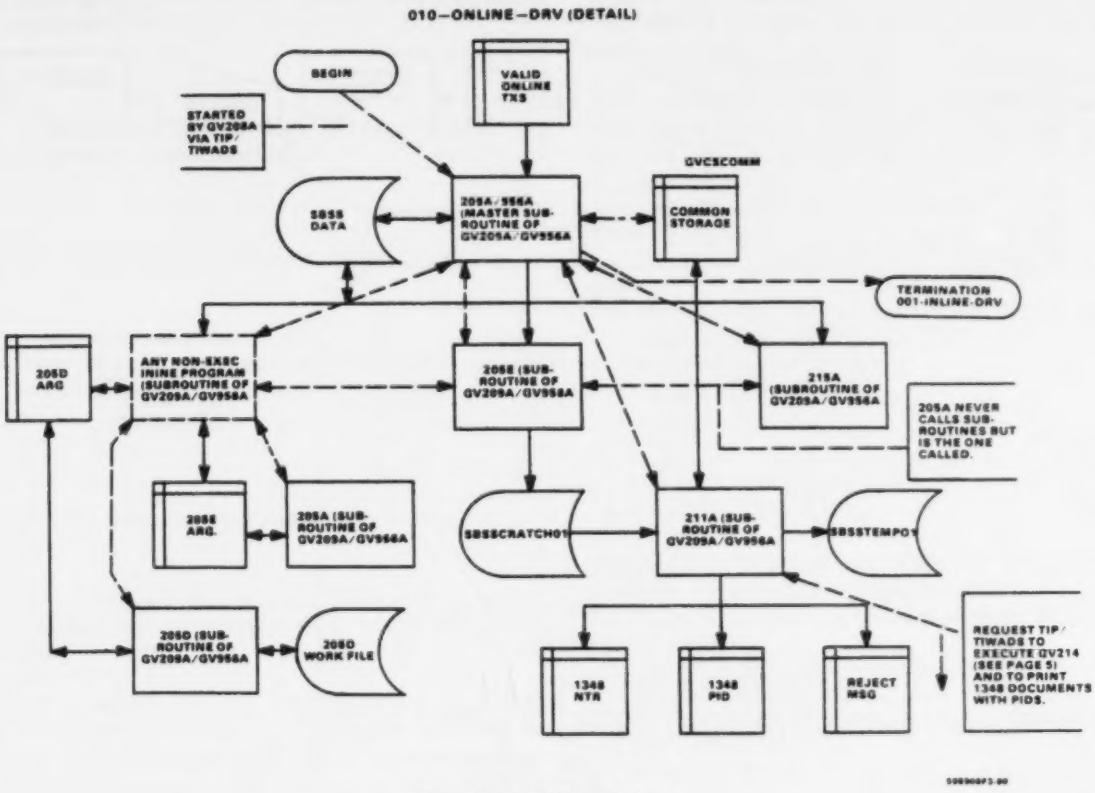

Figure 6–8. Logic flow of drivers.

subroutines or a fatal error occurred. If a fatal error (reject) terminated the TRIC processing, subroutine GV211A reads the reject data from SBSSCRATCH01, writes it to REJECT MSG, and calls TIP/TIWADS to display it on the input device. Also GV211A logs the input image, the reject information to DMR's audit trail tape, and returns control to the driver. GV211A will not distribute any of the other outputs (DD Form 1348-1 documents, etc.).

If the TRIC process is successful, GV211A logs the data saved by the online subroutines to the audit trail tape. Then it determines the distribution of the DD Form 1348-1 document by comparing the data from the online subroutine to the Base Constants-2 record. If the documents are headed for a PID device, then TIP/TIWADS is called to handle the output. If the document is to be output to an NTR device, the document is stored in a temporary work file for GV214. After these actions have been completed GV211A returns control to the online drivers (GV209A/GV956A).

Batch driver complex. To submit batch reports through the RPS–II card reader, the operator places an @RUN, @PASSWORD, and @XQT card for GV801A in front of the report select card(s) with an @FIN card following (fig. 6-9). This canned runstream may be input via the RPS–II card reader, UTS 40 (@GV801F), or host console or card reader. Whatever method of input is chosen, the end result is that the runstream is read into the system by the S1100 EXEC operating system and stored in the systems READ$ file. At this point the runstream is initiated and GV801A is executed. GV801A has five functions.

Function A of GV801A reads SBSS SELECTS, TRICS, and DATA which follow the '@XQT' of the canned runstream from the READ$ file. GV801A then writes them to the work files SELECTS, PARAMETERS, and TRICS to store them while it submits two '@ADD' ECL statements, which adds the 'SELECTS' and 'TRICS' elements from the master ECL file (USAF*GVECLUD001.) to the READ$ file. After all of this is accomplished, function B of GV801A takes over.

Function B of GV801A reads the 'SELECTS' and 'TRICS' elements from the READ$ file and builds the SELECT and TRIC tables. These tables are used to associate the name of the runstream elements on the master ECL to the SELECT or TRICS they will process. After the tables have been built, control is given to function C of GV801A.

Function C of GV801A reads the special control record from the SBSS data base to determine the SBSS processing mode (inline, twilight, reports, or utility) by checking the program control flag settings of flags A, C, and I. Next GV801A edits the TRIC and parameters for validity. If incorrect, the transaction is rejected and program GV898B is called to display the error (RPS MSG) to the RPS console

and return control to edit the remaining inputs in the READ$ file. The ECL runstreams are located from the MASTER ECL file (USAF*GVECLUD001.) and stored in an input file. After all ECL runstreams have been located and stored, control is given to function D of GV801A. Function *D* reads the master batch production scheduler (BPS) to identify the prerequisite records for the transactions to be processed. The master batch production schedule (BPS) records for each transaction are written to the local BPS file (nBPSrr*GV801AUD001.) changing the BPS key, which is the same as the master ECL element name to GVnnn, where nnn represents the number of selects or TRICs during this execution of GV801A (*Example:* the first select or TRIC will be assigned GV001, the second GV002, ..., etc.). The runstream stored by function C is then modified to perform the specific function required by each select or TRIC read. (See figures 6-5 and 6-6 for an example of before and after runstreams). Control is continually swapped between functions C and D until the runstreams for all the SELECTS and TRICS have been prepared. At that time function D gives control to function E of GV801A.

Function E passes the management number, restart number, and message to GV898B. GV898B displays the message to the RPS console and returns control to function E. GV801A calls GV801B to format and execute the '@START' control statement for each runstream created by this execution of GV801A. As each run is started GV801B updates the active flags of the Local BPS File. The execution of GV801A is terminated after all the runstreams created have been initiated.

As an analyst, supervisor, or operator, you must become aware of the processing relationships of the SBSS software. Be it system support, application, utilities or drivers, you must understand how they work together to process transactions and reports. Without this understanding you will find it difficult to analyze and identify software problems. If the software of the system doesn't work smoothly, neither will you.

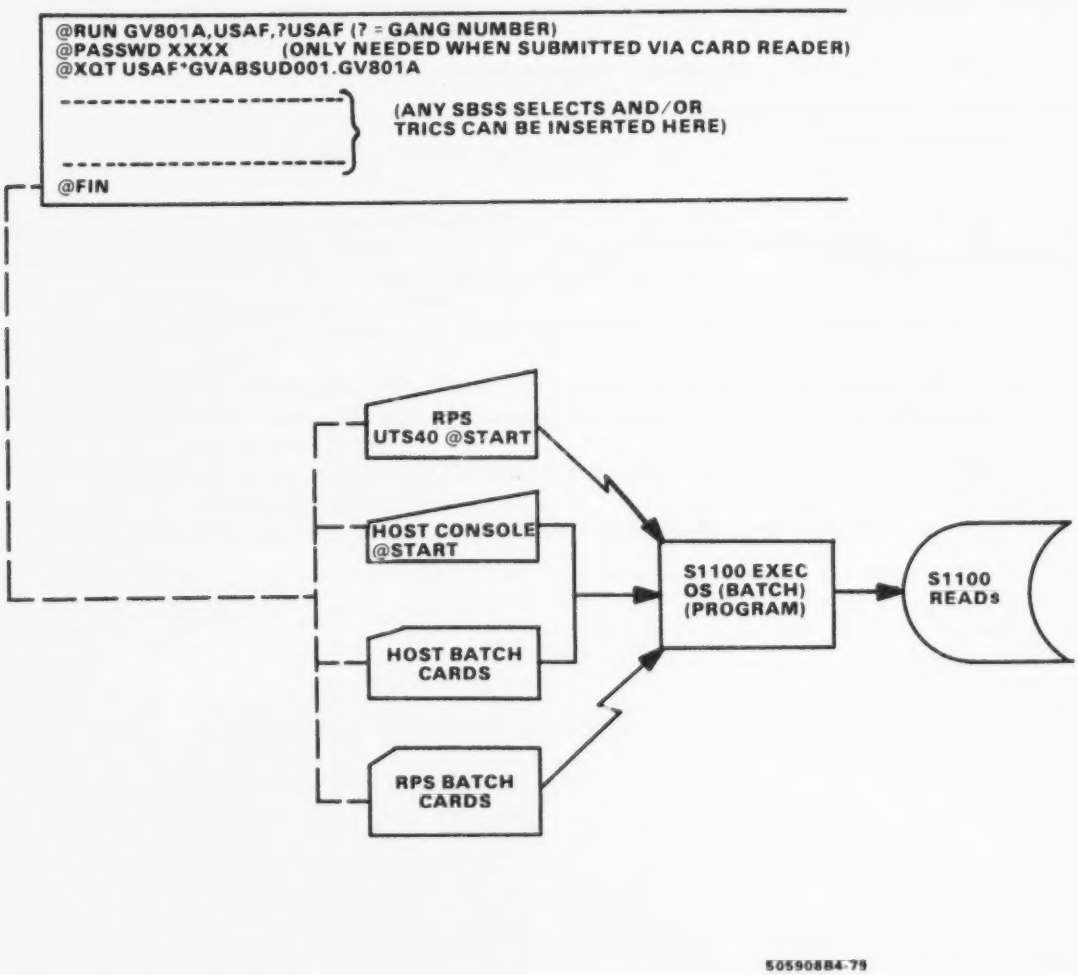

Figure 6–9. Canned runstream.

Exercises (232):

1. What are the two categories of drivers?

2. What programs control all input and output to and from TIP registered terminals?

3. What program is the functional system analyzer for SBSS?

4. What main subroutine processes the Accounting and Finance transactions?

5. What program is called by GV209A/GV956A to assign file SBSSCRATCH01 and log the input image?

6. What program is called by non-exec online subroutines to write their punch images?

7. What program is called by non-exec online subroutines to read data from a work file?

8. What program is called when a fatal error condition exists?

9. How many functions of GV801A are there?

10. What tables are built by function B of GV801A?

11. What does function C of GV801A determine by reading the special control record?

12. Function D of GV801A checks what file to identify prerequisite records for the transactions to be processed?

13. What program receives the management number, restart number, and message passed by function E of GV801A?

Bibliography

Department of Air Force Publications

AFM 67-1, Volume 2, *(Phase IV) Part Four, Automated Data Processing Machine Procedures*.

AFM 171-413, Volume 1, *Standard Base Supply Systems: D002A/GV Computer Operation Manual*.

Books

Executive Control Language, Handout KDA-431, 3390th Technical Training Group, Keesler AFB MS.

FURPUR, Student Handout KDA-421, 3390th Technical Training Group, Keesler AFB MS.

SBSS Transitioned Software Maintenance Programming Techniques, Sperry Corporation, 1983.

Commercial Manuals

UP-4144 *Executive System, EXEC Level 38R5A, Programmer Reference*, Sperry Corporation, 1983.

UP-7940 *Conversational Time Sharing (CTS) System Level 8R1, Programmer Reference*. Sperry Corporation, 1981.

UP-8117 *Introduction to Time Sharing for CTS Level 8R1 Users*, Sperry Coporation, 1981.

UP-8118 *Time Sharing Guide for CTS Level 8R1 Users*. Sperry Corporation, 1981.

UP-8225 *Sperry Univac 1100 Series Operating System; System Description*, Sperry Corporation, 1979.

UP-8723 *Text Editor (ED Processor) Level 16R1, Programmer Reference*. Sperry Corporation, 1980.

UP-8730 *System Utilities Programmer Reference*. Sperry Coporation 1982.

Glossary of Abbreviations

ADPE	Automated Data Processing Equipment
ADS	Automated Data System
ASCII	American Standard Code for Information Interface
CPU	Central Processing Unit
CSB	Computer Support Base
CTS	Conversational Time Sharing
DATA	Data Processor
DBMS	Data Base Management System
DDC	Data Definition Language
DML	Data Manipulation Language
DMR	Data Management Routine
DMS	Data Management System
DMU	Data Management Utilities
DPI	Data Processing Installation
DRU	Data Reorganization Utility
ECL	Executive Control Language
ELT	Element Processor
EXEC	Executive System
FURPUR	File Utility Routine/Program File Utility Routine
I/O	Input/Output
MFD	Master File Directory
QLP	Query Language Processor
SDDL	Subschema Data Definition Language
SECURE	File Administration Processor
TIP	Transaction Interface Processor
UTS	Universal Terminal System
VDU	Visual Display Unit

Answers for Exercises

CHAPTER 1

Reference:

200 –
1. The absolute quantity readily identified with each symbol in a numbering system.
2. Most significant.
3. One of a set of elementary symbols that may be arranged in ordered groups to express information.
4. Least significant.
5. a. Binary: 2; 0,1.
   b. Octal: 8; 0,1,2,3,4,5,6,7.
   c. Decimal: 10; 0,1,2,3,4,5,6,7,8,9.
   d. Hexadecimal: 16; 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F.
6. a. Binary system is the only numbering system actually used in the S1100/60. Data is stored in memory in an eight-bit character format.
   b. The octal numbering system eases the reading of binary numbers for operators and programmers.
   c. Data printed on listings and punched on cards are used and read in the decimal system.
   d. The hexadecimal system, as with the octal numbering system, eases the reading of binary values.
7. Four.
8. Two.

201 –
1. American Standard Code for Information Interchange.
2. 256.
3. Uppercase A–Z, 0–9, and special characters.
4. Other asset flag, data control flag, program control flag, and type level detail flag.

CHAPTER 2

202 –
1. Maintain overall system environment.
2. a. Runstream Processing. Verify the format of the runstreams, and control statements within. Once the runstream is accepted and all edits passed, the runstream is processed to completion.
   b. Resources/Facilities Allocation. Manage all system resources and allocate them to individual runs.
   c. Input/Output Handling. Perform all I/O operations for user programs.

203 –
1. (1) a.
   (2) f.
   (3) d.
   (4) e.
   (5) c.
   (6) b.
   (7) a.

204 –
1. a. File Control. Affords the highest degree of operation flexibility in storing and retrieving data.
   b. Facilities Assignment. Maintains current tables of the system's facilities.
2. System control, input/output, communications, and hardware confidence activities.
3. Coarse scheduler—schedules runs based on two factors: the priority of the run, and its facility requirements. Dynamic allocator—allots main storage space according to the needs of the individual tasks for each run. CPU dispatcher—switches the commitment of the CPU from one task to another.
4. Initiator—checks the status of hardware, selects the processing path, queues requests to the appropriate queue, and issues the I/O request to the specified device. Interrupt answerer—handles the interrupt, initial validation of the interrupt, queues the request for postprocessing, and issues the next I/O request.

205 –
1. Element processor.
2. Produce edited dumps of the contents of main storage at program termination. Providing programmers a research tool when a run is ERROR FIN'ed.
3. ARCHIVE.
4. Any collating sequence, fixed or variable.

CHAPTER 3

206 –
1. Label, operation, option, operand, and comment.
2. Operation.
3. Label.
4. Operand.
5. @.
6. Space.
7. @@ preceding the statement.

207 –
1. Qualifier, basic filename, F-cycle, read and write keys.
2. Provides uniqueness between files that have the same basic filename.
3. The external filename refers to the name of a cataloged file or a temporary file assigned to a particular run. An internal filename is a name attached to the external filename by the use of the @USE control statement.
4. F-cycles are used to identify a particular file from a set of related files that have the same qualifier and filename. Element-cycles specify a particular update of a symbolic element.
5. Element-name, version, and S-cycle.
6. Absolute, relocatable, symbolic, and omnibus.

208 –
1. @RUN.
2. @FIN.
3. @XQT.
4. @START.
5. @RUN D02,SBSS/LGSS,1USAF,,,1800.
6. @XQT GVABSUD001.PROG.

209 –
1. Names a file, states its I/O facility requirements, assigns it to a requesting run.
2. Used to catalog a file without having it assigned to the run.
3. Allows the user to release files and their facilities and removes the exclusive use of areas assigned to the run.
4. Equates filenames so that any particular file can be referenced by more than one filename.
5. Allows you to specify a standard qualifier for implied usage or succeeding control statements involving filenames.
6. @ASG,A USAF*GVABSUD001.
7. @CAT,P SBSS*RUNSTREAM//ETIRW.

210 –
1. @MSG.
2. @LOG.
3. @MSG,W PLZ RESPOND WITH THE TAPE NUMBERS OF THE LAST IRU DUMP.

211 –
1. Provides the means of printing a heading or successive pages of primary output along with the print files' cumulative page numbers and the current date when the primary output is being directed to a file.
2. Used to partition and redirect the primary output files PRINT$ and PUNCH$.
3. Directs the queuing of a previously created symbiont file to a specified symbiont device.
4. Permits the breakpointing and processing of files without the requirements associated with cataloging and assigning files when using the @BRKPT and @SYM control statements.
5. This control statement will print the header "RECEIPT CARD LIST" and suppress the printing of the date and page number.
6. This control statement directs the output of file SBSS *OUTPUT, to the device named NTR01P. It will print the file twice then decatalog the file SBSS*OUTPUT from the master file directory, due to the absence of the 'U' option.

212 – 1. @ADD.
2. @SETC, @TEST, and @JUMP.
3. T1, T2, S2, S3, or S4. Bit 30.
4. The control statement immediately following the @TEST.
5. Control is advanced to the control statement.
6. a. @SETC,A S4/10.
b. @TEST TEP/S4.
c. @JUMP 5.

213 – 1. @DATA.
2. @ELT.
3. @END or @FIN.
4. Used to mark the end of a data file or element.
5. It specifies that images to be loaded to the file are immediately behind the @DATA control statement.
6. It is used to specify that control statements are to be input as symbolic data.

214 – 1. (1) a.
(2) e.
(3) d.
(4) b.
(5) c.
2. @PRT,T SBSS*FILE.
3. The write key for the file SBSS*INLINE is being changed from 'key' to 'lock'.
4. @CHG,P SBSS*FILEC//8246.,SBSS*FILEC//ACFG.
5. @PRT,TL SBSS*PROGRAMS.

CHAPTER 4

215 – 1. (1) j.
(2) i.
(3) h.
(4) f.
(5) g.
(6) c.
(7) d.
(8) e.
(9) b.
(10) a.

216 – 1. I and N. 'I' causes a full initialization process complete with checking the news file and printing an online assistance reminder. The 'N' option also causes a full initialization but does not check the news file nor print an online assistance reminder.
2. CTS$FILE.
3. P–fieldata and Q–ASCII.
4. Through signing on CTS without any options in the control statement or by initializing CTS after a system crash.
5. XCTS.

217 – 1. (1) d.
(2) c.
(3) b.
(4) a.
(5) g.
(6) f.
(7) e.

218 – 1. (1) d.
(2) c.
(3) b.
(4) a.
(5) f.
(6) i.
(7) h.
(8) e.

219 – 1. (1) d.
(2) f.
(3) h.
(4) i.
(5) g.
(6) a.
(7) e.
(8) c.
(9) b.

220 – 1. RELEASE.
2. N.
3. To associate an interrupt filename to an assigned external filename.
4. The file will be empty, yet it retains its identity. If it was identified as a program file or data file on the master file directory, it will still be identified as such after it has been erased.
5. RELEASE SBSS*REPORTS.
6. PACK,RI USAF*PROGRAMS.

221 – 1. LIST INUSE.
2. Lists the elements saved last, first.
3. D.
4. Prints the number of lines of data in the working area 'f,' and the line number of the last line in 'f.'
5. MAIL.
6. HELP.
7. LIST SAVED,SD

CHAPTER 5

222 – 1. Call the text editor processor and specify the input, output, and operational modes of the processor.
2. All lines to be inserted directly into the text. Used initially to build files and runstreams.
3. Used to update an existing file on element, it allows various commands to be used to modify the text.
4. Enter a blank line or by entering @Edit.
5. Q.
6. R.

223 – 1. ADD.
2. APPEND.
3. Used to make corrections in the text by replacing the contents of string-1 with the contents of string-2.
4. DELETE.
5. DITTO.
6. INSERT.
7. MOVE.
8. RETYPE.
9. CHANGE /XZY/XYZ/5 ALL
10. DELETE 25 45
11. DITTO 75 100

224 – 1. Search the text for an image which corresponds exactly, column for column starting at column one with the mask of the command. All occurrences of the image in the rest of the text are printed.
2. Similar to the FC command except it prints only the first occurrence.
3. It searches the text for specific occurrences of an image. It differs from the FC command by not using a mask. It is not position oriented.
4. Similar to the LC command except it prints only the first occurrences of the string.
5. FC*5 ABCD.

6. LOCATE. RPTD02.

225 – 1. Specifies that an automatic save of the current file is to be performed. You may specify the frequency the auto save will perform. This frequency is the number of inputs before the file is saved.
2. EXIT.
3. Positions the editor at a line numbered according to the input text, even after the input file has been edited.
4. LAST.
5. Used to position the editor at a desired line in the text.
6. INPUT.
7. Ignore all changes made to the file and exit Text Editor.
8. SET.
9. TAB!

226 – 1. OPR*.
2. PRINT.
3. SITE.
4. MSCHAR !.
5. IB.
6. NEXT.
7. IB.
8. SCALE 25 70

CHAPTER 6

227 – 1. System Support, Application, Utility, Driver.
2. Drivers.
3. Utility.

228 – 1. Serves to control SBSS-related documentation and software distribution.
2. 5.

229 – 1. U-unclassified.
2. GV818.
3. 8.
4. Disk.

230 – 1. Contains the tables and runstreams used by SBSS drivers to select the proper application program and ECL runstream to process SBSS transactions.
2. USAF*GVECLUD001.SELECTS.
3. The runstream created by GV801A to process SBSS transactions.
4. Controlling runs in the SBSS/ADS batch environment.
5. USAF*GVABSUD001., SBSS*DBALIB$., DMS*DBALIB$., TIP$*TIPABS$., and GVCMP*ABS.

231 – 1. GV208A.
2. Used to output documents to specified batch devices.
3. GV801B.
4. Acts as a receiver for a response asked the RPS console operation and passes the response to the application program.
5. Serves as communication link between the programs required to process SBSS A/F transactions.
6. GV208B.
7. GV211A/B.
8. GV215.
9. GV801C.
10. GV801D.
11. GV801E.
12. GV898C.

232 – 1. Online and batch.
2. TIP/TIWADS.
3. GV208A.
4. GV956A.
5. GV205B.
6. GV205A.
7. GV205D.
8. GV215.
9. Five.
10. SELECT and TRIC.
11. The SBSS processing mode.
12. Master batch production schedule file.
13. GV898B.

AUGAFS, AL (582490) 1100
a31-026 / 40258